Р. Хоторн

Разработка баз данных Microsoft SQL Server™ 2000 на примерах

СЕРИЯ ПРОГРАММИРОВАНИЯ

ДЖЕССА ЛИБЕРТИ

на примерах

Разработка баз данных Microsoft® SQL Server™ 2000

на примерах

Изучение процесса разработки базы данных SQL Server 2000 на примере создания приложения SpyNet







Роб Хоторн



Разработка баз данных Microsoft® SQL Server™ 2000



Роб Хоторн



Издательский дом "Вильямс"
Москва · Санкт-Петербург · Киев
2001

ББК 32.973.26-018.2.75
Х85
УДК 681.3.07

Издательский дом "Вильямс"

Перевод с английского *В.В. Александрова, А.В. Журавлева* и канд. техн. наук *А.Г. Сысонюка*

Под редакцией *В.В. Александрова* и *А.В. Журавлева*

Общая редакция *В.В. Александрова*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу: info@williamspublishing.com, http://www.williamspublishing.com

Хоторн, Роб.

Х85 Разработка Microsoft SQL Server 2000 на примерах. : Пер. с англ. — М. : Издательский дом "Вильямс", 2001. — 464 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0187-1 (рус.)

Книга, которую вы держите в руках, отличается от всех книг по SQL Server 2000, издававшихся когда-либо ранее. Вспомните стандартную схему подачи материала в книгах, посвященных разработке программного обеспечения: сначала предлагается наиболее простая информация, направленная на приобретение элементарных навыков, затем обсуждаются более сложные вопросы и наконец рассматривается какая-либо демонстрационная программа-пример, создание которой предполагает использование всего изученного материала. Данная книга начинается не с изучения методики программирования (хотя в ней представлен материал, посвященный теории баз данных), а непосредственно с самого проекта, что предполагает его тщательный анализ и разработку основных требований, в соответствии с которыми проект "проводится в жизнь".

Книга предназначена для подготовленных пользователей.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Que Corporation.

Authorized translation from the English language edition published by Macmillan Computer Publishing Copyright © 2001

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2001

ISBN 5-8459-0187-1 (рус.)
ISBN 0-7897-2447-2 (англ.)

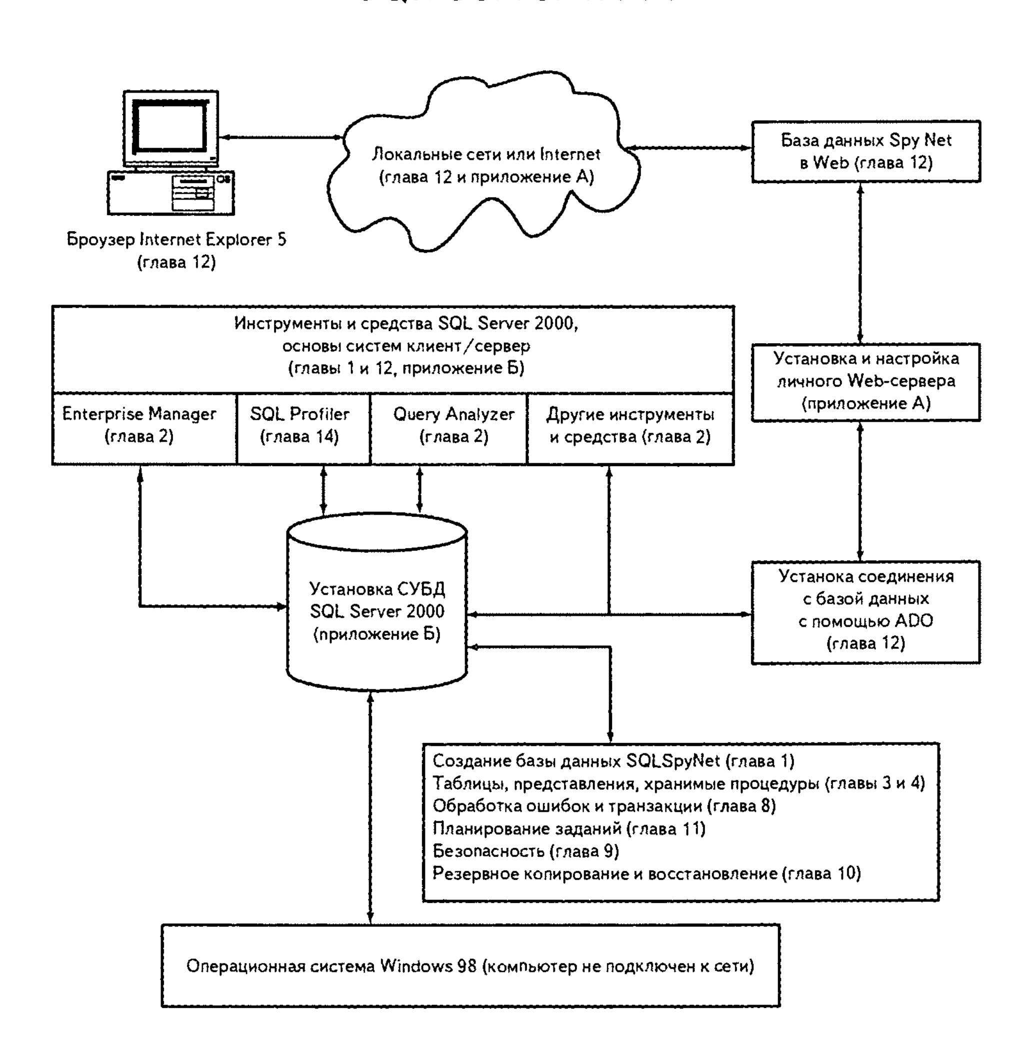
© Издательский дом "Вильямс", 2001
© Que Corporation, 2001

Оглавление

Содержание

Разработка баз данных Microsoft SQL Server 2000 на примерах

Взаимосвязь между концепциями и технологиями разработки приложений базы данных Microsoft SQL Server 2000



Об авторе

Роб Хоторн (Rob Hawthorne) работает в подразделении интеграции электронных решений (eIntegration) компании KPMG Consulting, базирующейся в Веллингтоне, Новая Зеландия. Кроме этого, Роб тесно сотрудничает с подразделением стратегии и развития электронных решений (eStrategy and Process) той же компании. Ранее Роб работал в подразделении решений для электронной коммерции (E-Commerce) компании Advantage Group Limited, где занимался разработкой реальных приложений для "настоящих" деловых людей.

Для того чтобы получить дополнительную информацию о компаниях KPMG Consulting и Advantage Group Limited, посетите их Web-узлы в Internet по адресам: `http://www.kpmgconsulting.com` и `http://www.advantagegroup.co.nz`.

Роб Хоторн окончил университет Отаго (University of Otago — `http://www.otago.ac.nz`), который считает лучшим университетом в мире; его профилирующими специальностями были информационные технологии и операционный менеджмент.

Одним из важнейших приоритетов Роб считает совершенствование знаний, а потому, помимо всего прочего, он получил сертификаты специалиста Microsoft — MCP и MCSD. И если бы не эта книга, сделавшая его на несколько недель очень занятым человеком, он наверняка бы успел подготовиться и сдать еще несколько квалификационных экзаменов!

Роб живет в столице Новой Зеландии Веллингтоне с любящей женой и великолепными детьми (его адрес: `http://www.wellingtonnz.com`). Свободное от работы время Роб посвящает прогулкам на горном велосипеде, наслаждаясь прекрасными пейзажами новозеландской глубинки.

Новая Зеландия — живописнейшая страна. Неискушенный человек приходит в восторг от ее белоснежных горных вершин, тихих озер и залитых солнцем пляжей. Здесь нет ядовитых змей, пауков и прочих животных, таящих опасность для человека (ну, парочку змей и пауков все же найти можно, но они совсем не ядовиты!). Новая Зеландия известна и как отличное место для занятий горными лыжами и сноубордингом. А где еще вы найдете такие роскошные зеленые луга и пастбища?! Именно потому Роб так часто повторяет: "Я просто обожаю эту страну!". Для того чтобы получить дополнительную информацию о Новой Зеландии, посетите Web-узел в Internet по адресу: `http://www.purenz.com`.

Как вы уже поняли, Роб очень гордится своей страной. Помимо всего прочего, Новая Зеландия имеет такую великую команду, как All Blacks[1]. И хотя последний чемпионат мира не выигран, кубок Америки все еще за нами!

---

[1] *Одна из самых известных и экстравагантных команд по регби в мире.* — Прим. ред.

Посвящение

Я хочу посвятить эту книгу моей маме Дафни Скиннер (Daphne Skinner). Мама всегда помогала мне своей поддержкой, советом и вдохновляла на успех не только во время работы над этой книгой, но и во многих других начинаниях.

К сожалению, мама так и не дождалась публикации книги, оставив нас во время ее написания. Но я знаю, что мама видит и слышит меня, а потому хочу сказать: "Спасибо, мама! Ты дала мне очень много!

Спасибо тебе за любовь, нежность, постоянную поддержку, а также за то мужество, которое ты принесла в этот непростой мир и которым ты делилась со мной в трудные минуты.

Ты навсегда останешься в моем сердце!"

Благодарности

Я хотел бы поблагодарить всех тех людей, которые оказывали мне помощь, поддержку и вдохновляли меня во время работы над книгой.

Прежде всего, выражаю огромную благодарность команде из издательства Que Publishing: ребята, вы были просто великолепны! Особо хотелось бы отметить следующих сотрудников издательства.

- Холли Аллендера (Holly Allender) за предоставленную возможность написать эту книгу. Без Холли ничего бы не получилось. Спасибо тебе, Холли.
- Эдил Рихэн (Adil Rehan) и Джима Купера (Jim Cooper) за те неоценимые советы, которые они давали мне в минуты, когда я отклонялся от намеченного курса. Ваша блестящая техническая подкованность очень помогла мне. Спасибо, друзья.
- Мишель Ньюкомб (Michelle Newcomb) за проявленное ею терпение и понимание. И хотя работа над книгой далеко не всегда оказывалась такой уж безоблачной, Мишель была просто великолепна, отвечая на мои глупые вопросы и сохраняя при этом терпение и желание помочь. Спасибо тебе, Мишель.
- Марлу Рис-Холл (Marla Reece-Hall) за ее удивительные педагогические способности. Ты была строгой и в то же время в высшей мере справедливой. Спасибо тебе, Марла, за помощь и замечательные советы.
- Тоню Симпсон (Tonya Simpson) за умение удерживать мой строптивый новозеландский характер в рамках приличия, что в конечном итоге помогло этой книге попасть на полки магазинов. Спасибо тебе, Тоня, за то, что ты, наряду с Мишель и Марлой, сделала возможным само появление этой книги.

Хочу поблагодарить профессоров университета Отаго за обучение меня тем фундаментальным дисциплинам, знание которых позволило мне написать эту книгу (а также, между прочим, сделать неплохую карьеру): д-ра Стивена МакДонелла (Stephen MacDonell), д-ра Джоффа Кеннеди (Geoff Kennedy), Дэйва Кэмпбелла (Dave Campbell) и д-ра Ричарда Паско (Richard Pascoe). Вы положили начало моей карьере и предопределили мой глубокий интерес в области разработки программного обеспечения. Спасибо!

Я благодарен всем своим коллегам по работе за понимание того, какую огромную дополнительную нагрузку я взял на себя в связи с написанием этой книги. Отдельное спасибо Тони Стюарту (Tony Stewart), Пэдди Пэйн (Paddy Payne) и команде Evolve за полное понимание и поддержку.

Выражаю также огромную признательность двум очаровательным женщинам.

- Ким Танзельман (Kim Tunzelman) за ее кропотливый труд. Без тебя, Ким, у нас не было бы тех удивительных имен тайных агентов и "плохих парней", которые ты подобрала со свойственной тебе тщательностью. Спасибо, Ким, ты была великолепна!
- Джиллиан Ки (Jillian Key) за замечательную графику. Без тебя, Джиллиан, дизайн Web-узла SQLSpyNet получился бы скучным и неинтересным. Ты была просто непревзойденной!

И наконец, я очень благодарен своей жене и детям за терпение и поддержку. Я люблю вас, более того — горжусь вами! Обещаю, Джейкоб, что с этого дня папа бу-

дет каждый вечер читать тебе ровно по две сказки. Спасибо, сынок, что все это время ты был таким терпеливым.

Несмотря на то что я уже выразил благодарность всем, кто помогал мне делать эту книгу, хочу еще раз подчеркнуть, что никакие "спасибо" не способны выразить тех чувств, которые я испытываю по отношению к каждому из вас. Вы все были просто великолепны, и без вас я вряд ли бы смог достичь того, чего достиг...

Очень надеюсь, что в ближайшем будущем у меня еще будет шанс поработать с каждым из вас.

Роб Хоторн

Введение

Книга, которую вы держите в руках, отличается от всех *когда-либо ранее* издававшихся книг по SQL Server 2000. Вспомните стандартную схему подачи материала в книгах, посвященных разработке программного обеспечения: сначала изучаются простые аспекты, направленные на приобретение элементарных навыков, затем все более и более сложные, и наконец в финале книги рассматривается какая-либо демонстрационная программа-пример, создание которой предполагает использование всего изученного материала.

Данная книга начинается не с изучения методики программирования (хотя, естественно, в ней представлен материал, посвященный теории баз данных), а непосредственно с самого проекта, что предполагает его тщательный анализ и разработку основных требований, в соответствии с которыми проект "проводится в жизнь". Таким образом, навыки разработчика приобретаются в контексте создания проекта; поэтому главную идею книги можно сформулировать так: "Сначала пойми, что ты хочешь сделать, а уж затем изучи необходимые для этого технологии".

На кого рассчитана эта книга

Прежде всего эта книга будет полезна следующим категориям читателей:

- разработчикам программного обеспечения, желающим расширить свои навыки и изучить различные аспекты управления базами данных;
- разработчикам Microsoft Access, которые почувствовали необходимость создания более масштабных приложений;
- Web-дизайнерам (HTML-кодировщикам), которые желают разрабатывать динамические приложения, взаимодействующие с базами данных;
- программистам C/C++ (а также использующим другие языки программирования), которые хотят переключиться на разработку и управление базами данных;
- новичкам в области разработки программного обеспечения и систем управления базами данных (СУБД), которые хотят приобрести необходимые для продвижения по службе или получения простого морального удовлетворения знания.

Синтаксис языка SQL (т.е. ключевые слова, используемые при создании запросов) несложен. Можно сказать, что он наиболее приближен к английскому языку, нежели любой другой компьютерный язык. Несмотря на это, при практическом использовании SQL наиболее важно сконцентрироваться не на синтаксисе этого языка, а на его *семантике* (т.е. на тех действиях, которые предполагается совершать с его помощью).



Синтаксис языка — это правильное использование его терминов и пунктуации. Аналогом синтаксиса компьютерного языка выступает грамматика английского языка (это касается структуры предложения).

Термин *Семантика* — это смысл и предназначение компьютерного кода. Семантику можно сравнить с понятием или идеей, которую вы хотите выразить и донести до других людей. Например, при объяснении какой-либо идеи можно воспользоваться не только голосом (или словами), но еще и жестикуляцией или наглядными пособиями (рисунками).

Соглашения, принятые в книге

Ниже приведены некоторые из основных особенностей, используемые в книгах серии *...на примерах*.

Термин *Термин* — данная пиктограмма обозначает использование нового термина. Всякий новый термин выделяется в абзаце текста *курсивом*.

Код для запуска → Данная пиктограмма располагается рядом с программным кодом, который необходимо ввести, откомпилировать и выполнить.

Экскурс Экскурсы позволяют несколько отклониться от обсуждаемого вопроса с тем, чтобы лучше усвоить суть изучаемого материала.

Учитывая специфический тип книги, некоторые вопросы могут рассматриваться в различных ее частях, в зависимости от того, когда и где в процесс разработки приложения была добавлена та или иная особенность. Для того чтобы прояснить принцип взаимодействия всех частей приложения, в начале книги размещается *концепция Web-приложения* — схематический рисунок, объясняющий принцип взаимоотношения используемых при разработке приложения программных концепций.

На заметку В этих вставках вы найдете много полезных комментариев и связанных с рассматриваемой темой отступлений. Помимо этого, здесь часто приводятся исчерпывающие объяснения определенных концепций.



В этих вставках размещаются подсказки и полезные рекомендации, призванные повысить эффективность разработки приложений на основе SQL Server 2000.



В этих вставках перечисляются самые распространенные ошибки программирования, которые могут испортить вам не один выходной день и существенно замедлить процесс разработки приложения.

Для выделения особо важных фрагментов текста в книге принято следующее шрифтовое форматирование:

- команды, переменные и другой программный код выделяются специальным `моноширинным шрифтом`;

- одним из принципов данной книги является повторное использование ранее созданного программного кода. Добавляемые в существующий код новые фрагменты выделяются **моноширинным полужирным шрифтом**;
- команды и текст, который вам предлагается ввести, выделяется **полужирным шрифтом**;
- некоторый текст в описании синтаксиса команды выделяется *моноширинным курсивом*. Это означает, что вместо указанного текста необходимо подставить свой собственный (например, имя файла, значение параметра и т.д.).

Способ представления программного кода

Можно заметить, что в некоторых случаях встречающийся в книге программный код представлен несколько необычным образом: отдельные его строки разбиты на две части, а номера строк включают в себя буквы. Для примера рассмотрим строки 10 и 10а:

```
10:  SELECT Column1, Column2, Column3, Column4
10a: FROM Table1 WHERE Column1 = 512
```

Легко заметить, что это одна и та же строка программного кода, которая из-за большой длины разбита на две части (что позволило ей разместиться на странице данной книги). Несмотря на это, представленный в строке код пригоден к использованию и может быть введен именно в том виде, в котором был представлен на странице книги (естественно, при этом необходимо опустить номера строк).

Расположенная в номере строки буква *а* указывает на то, что в обычной ситуации эти две строки кода должны быть объединены в одну. Например, рассмотренные выше строки могут быть объединены в одну длинную строку:

```
10:  SELECT Column1, Column2, Column3, Column4 FROM Table1 WHERE Column1 = 512
```

В большинстве случаев, чтобы разделенная на части строка кода была пригодна к использованию, в нее следует внести некоторые коррективы. Для примера рассмотрим следующую строку:

```
10:  SELECT @MyVar = 'A string value in the variable for demonstration'
```

Для того чтобы разбить эту строку на две части, необходимо завершить выражение SELECT и разместить еще одно такое выражение на следующей строке:

```
10: SELECT @MyVar = 'A string value in the variable'
10a:SELECT @MyVar = @MyVar + 'for demonstration'
```

Убрав номера строк, мы опять-таки получим вполне корректное с точки зрения синтаксиса языка SQL выражение, хотя оно и будет несколько отличаться от своего первоначального вида. (Не волнуйтесь, если вы ничего не поняли из рассмотренного до сих пор кода; все, что связано с языком SQL, подробно объясняется в последующих главах книги.)

Что рассматривается в данной книге

Эта книга построена таким образом, чтобы помочь читателю получить как можно больше практических навыков работы с SQL Server 2000. Разрабатываемое в ней приложение создается "с нуля", начиная со стадии анализа и проектирования и заканчивая стадией разработки интерфейса пользователя.

В процессе разработки приложения вы приобретете ценный опыт и получите знания, касающиеся языка SQL Server 2000, а также многих предлагаемых SQL Server 2000 средств.

Необходимо отметить, что эта книга не претендует на звание "наиболее исчерпывающего источника информации" среди всех книг, посвященных SQL Server 2000. Подобные амбиции свойственны лишь таким глобальным информационным ресурсам, как Microsoft Developer Network (MSDN). Предназначение данной книги — наглядная иллюстрация всех стадий процесса создания приложения, использующего в своей работе базу данных, и управления им. На основе различных примеров книга помогает получить навыки администрирования и поддержки подобного приложения.

Приложение Б, "Установка и настройка SQL Server 2000", этой книги содержит руководство по установке SQL Server 2000 на компьютер под управлением операционной системы Windows 98. Там же можно найти и советы по установке SQL Server 2000 на другие платформы.

Версии SQL Server 2000

SQL Server 2000 поставляется в трех основных и трех облегченных версиях.

Версия *Personal Edition* предназначена для установки на компьютеры под управлением Windows 98, Windows NT 4.0 Workstation и Windows 2000 Professional. База данных приложения, созданная с помощью этой версии SQL Server 2000, может быть оптимизирована для использования на отключенном от сети или мобильном компьютере. Кроме этого, версия *Personal Edition* идеально подходит для разработки небольших приложений, ориентированных на взаимодействие с базой данных, которые как правило запускаются на отдельном компьютере. Учитывая перечисленные выше особенности, можно отметить, что именно эта версия наиболее подходит для того, чтобы изучить основные средства SQL Server 2000, не вдаваясь в сложные вопросы, связанные с *многопользовательской* средой.

Термин Термин *многопользовательский* обозначает возможность одновременной работы с приложением нескольких пользователей. Они могут выполнять какую-либо одну задачу, разные задачи или же вообще ничего не делать, оставаясь в то же время подключенными к приложению. На стадии разработки программного продукта вопросу поддержки многопользовательского окружения необходимо уделить особое внимание.

Следующей версией SQL Server 2000 является *Standard Edition*. Как правило, она используется одним разработчиком (или небольшой группой) для объединения и получения более широких функциональных возможностей. В некотором смысле *Standard Edition* — дешевая альтернатива версии *Enterprise Edition*.

Наиболее полный пакет SQL Server 2000 поставляется с версией *Enterprise Edition*, которая предназначена для использования этого приложения в масштабах крупного предприятия. *Enterprise Edition* поддерживает *отказоустойчивую кластеризацию* серверов SQL Server 2000, что позволяет обрабатывать сотни тысяч строк информации и обеспечивать одновременную работу сотен пользователей. Будучи довольно сложным, описание процедур установки и администрирования этой версии SQL Server 2000 выходит за рамки данной книги.

Термин *Отказоустойчивая кластеризация* подразумевает использование нескольких серверов SQL Server 2000 для поддержки работоспособности системы на случай аварийного сбоя или запланированных профилактических отключений. Если по каким-либо причинам один из входящих в кластер серверов перестанет функционировать, его обязанности примет на себя другой сервер (серверы), входящий в кластер. Таким образом, отказоустойчивая кластеризация позволяет добиться высокой работоспособности и доступности приложения.

Ниже перечислены три облегченные версии SQL Server 2000, которые могут понадобиться вам в ближайшем будущем.

Версия *Developer Edition* предназначена для разработчиков, занятых созданием полномасштабного приложения, использующего SQL Server 2000. *Developer Edition* включает в себя все особенности и преимущества версии *Enterprise Edition*, однако имеет лицензию, разрешающую использование этой версии SQL Server 2000 только для разработки и тестирования приложений. Таким образом, *Developer Edition* нельзя использовать в качестве основы для построения полномасштабной системы уровня крупного предприятия.

Desktop Engine — это версия процессора базы данных SQL Server 2000, не имеющая ограничений на свободное распространение (так называемая исполняемая версия). Отдельные разработчики могут использовать эту версию при распространении собственных приложений. Естественно, что *Desktop Engine* лишена большинства возможностей, присущих другим версиям SQL Server 2000. Используя *Desktop Engine*, разработчики приложений могут создавать программы с интерфейсом, созданным с помощью Microsoft Visual Basic (VB) или любого другого подобного средства, и базой данных, использующей процессор SQL Server 2000.

Версия *CE Edition* предназначена для использования на устройствах, работающих под управлением операционной системы Windows CE. Она позволяет хранить и извлекать информацию из базы данных и может быть использована для синхронизации с SQL Server 2000 при возвращении на постоянное рабочее место.

Так для чего же все-таки предназначена данная книга

Материал в этой книге представлен таким образом, чтобы помочь читателю основательно подготовиться к разработке сложных приложений:

- развить навыки, необходимые для эффективной разработки приложения, взаимодействующего с базой данных, на примере приложения SQLSpyNet (Сеть тайных агентов);
- усвоить некоторые профессиональные подходы к решению проблем, базирующиеся на промышленных стандартах, знание которых может сослужить хорошую службу в будущем;
- научиться использовать математический аппарат, в особенности теорию множеств;
- получить представление о возможных направлениях совершенствования знаний, касающихся разработки взаимодействующих с базами данных приложений;
- заложить крепкий фундамент, на основе которого можно будет продолжить совершенствование навыков в области управления SQL Server 2000;
- научиться устанавливать SQL Server 2000 на компьютер под управлением операционной системы Windows 98.

На заметку В этом месте наиболее любознательные читатели не без основания могут задать вопрос: если SQL Server 2000 может работать на компьютере под управлением Windows 2000 Professional, почему мы устанавливаем его на платформу Windows 98? Самая главная причина в том, что SQL Server 2000 практически одинаково работает под управлением как операционной системы Windows 2000, так и Windows 98. Однако это ни в коем случае не означает полного отсутствия различий, просто они столь несущественны, что, пожалуй, не заслуживают хоть сколько-нибудь пристального внимания.

При написании данной книги одним из основных требований было ее ориентирование на "среднестатистического" читателя (который, согласитесь, скорее всего установит SQL Server 2000 именно на компьютер под управлением Windows 98). Тем не менее везде, где только можно, была предпринята попытка описать различия между функционированием SQL Server 2000 под управлением Windows 2000 Professional и Windows 98, чтобы помочь читателю избежать каких-либо недоразумений и неудобств во время работы.

И все-таки следует еще раз отметить, что главное отличие между SQL Server 2000, функционирующими под управлением операционных систем Windows 2000 Professional и Windows 98 (или даже NT 4.0), заключается в некотором (причем только возможном) отличии внешнего вида диалоговых окон. Таким образом, вы можете разрабатывать приложение и продолжать работу над проектом, не задумываясь о возможных проблемах.

В следующем разделе рассматривается несколько важных вопросов, связанных с системными требованиями, предъявляемыми к SQL Server 2000.

Требования, предъявляемые к аппаратному и программному обеспечению

Рассмотрим требования, предъявляемые к аппаратному и программному обеспечению при установке SQL Server 2000. Следует отметить, что они не являются слишком "вычурными", однако их полезно узнать еще до того, как вы начнете устанавливать SQL Server 2000 с компакт-диска.

Прежде всего рассмотрим требования, предъявляемые к аппаратному обеспечению, после чего перейдем к программному обеспечению, отмечая базовые требования, касающиеся различных вариантов установки SQL Server 2000.

Требования, предъявляемые к аппаратному обеспечению при установке SQL Server 2000

Опишем требования, предъявляемые при установке SQL Server 2000 к аппаратному обеспечению. (Детальное руководство по его установке представлено в приложении Б, "Установка и настройка SQL Server 2000".) Ниже приведен составленный Microsoft список требований к аппаратному обеспечению, которые предъявляются при установке SQL Server 2000:

- тактовая частота процессора не менее 166 МГц;
- минимум 64 Мбайт оперативной памяти при установке версии Enterprise Edition и 32 Мбайт при установке любой другой версии;
- 180 Мбайт дискового пространства при полной установке;
- 170 Мбайт дискового пространства при стандартной установке;
- 65 Мбайт дискового пространства при минимальной установке;
- для установки дополнительного программного обеспечения Analysis Services потребуется еще 50 Мбайт дискового пространства (обратите внимание, что установка этого средства SQL Server 2000 необязательна);
- для установки дополнительного программного обеспечения English Software вам потребуется еще 40 Мбайт дискового пространства (установка этого средства SQL Server 2000 также необязательна).

Хотя, как было указано выше, Microsoft рекомендует устанавливать SQL Server 2000 на компьютер с минимальной тактовой частотой процессора 166 МГц, при написании этой книги копия SQL Server 2000 Personal Edition была установлена на компьютер, имеющий 32 Мбайт оперативной памяти, 1 Гбайт дискового пространства и процессор Pentium с тактовой частотой 75 МГц.

Таким образом, Microsoft выпустила приложение, которое может быть установлено не в соответствии с предъявляемыми к нему минимальными требованиями (за исключением объема оперативной памяти и дискового пространства), а в соответствии с минимальными требованиями, предъявляемыми к операционной системе, управляющей компьютером, на который будет установлено это приложение. Вследствие этого даже самые упорные консерваторы, годами не желающие обновлять свои компьютеры, могут ознакомиться с наиболее современным программным обеспечением. (Бурные, продолжительные аплодисменты!)

Требования к аппаратному обеспечению, предъявляемые при установке других версий SQL Server 2000

Данное руководство по выбору аппаратного обеспечения подходит для большинства версий SQL Server 2000; исключение составляет лишь Enterprise Edition, которая, как отмечалось выше, требует минимум 64 Мбайт оперативной памяти.

При установке SQL Server 2000 Enterprise Edition на кластер компьютеров, входящих в большую корпоративную сеть (сеть крупного предприятия), накладываются ограничения, касающиеся максимального объема оперативной памяти (64 Гбайт) и количества процессоров (32) при использовании операционной системы Microsoft Windows 2000 DataCenter. И хотя для большинства читателей этой книги приведенные выше параметры все еще относятся к области научной фантастики, многим организациям требуются именно такие, а порой и более совершенные аппаратные средства для обеспечения доступности и работоспособности приложений в течение 24 часов в сутки и семи дней в неделю. Стоит отметить, что именно в компьютерных сетях, оснащенных достаточно мощным аппаратным обеспечением, становится возможным создание отказоустойчивых и надежных систем.

На этом обзор аппаратного обеспечения, необходимого для установки SQL Server 2000 на выбранную платформу, можно считать завершенным. Теперь осталось только убедиться, что ваша система удовлетворяет всем требованиям, предъявляемым к программному обеспечению, необходимому для установки SQL Server 2000, после чего можно приступать непосредственно к разработке приложения.

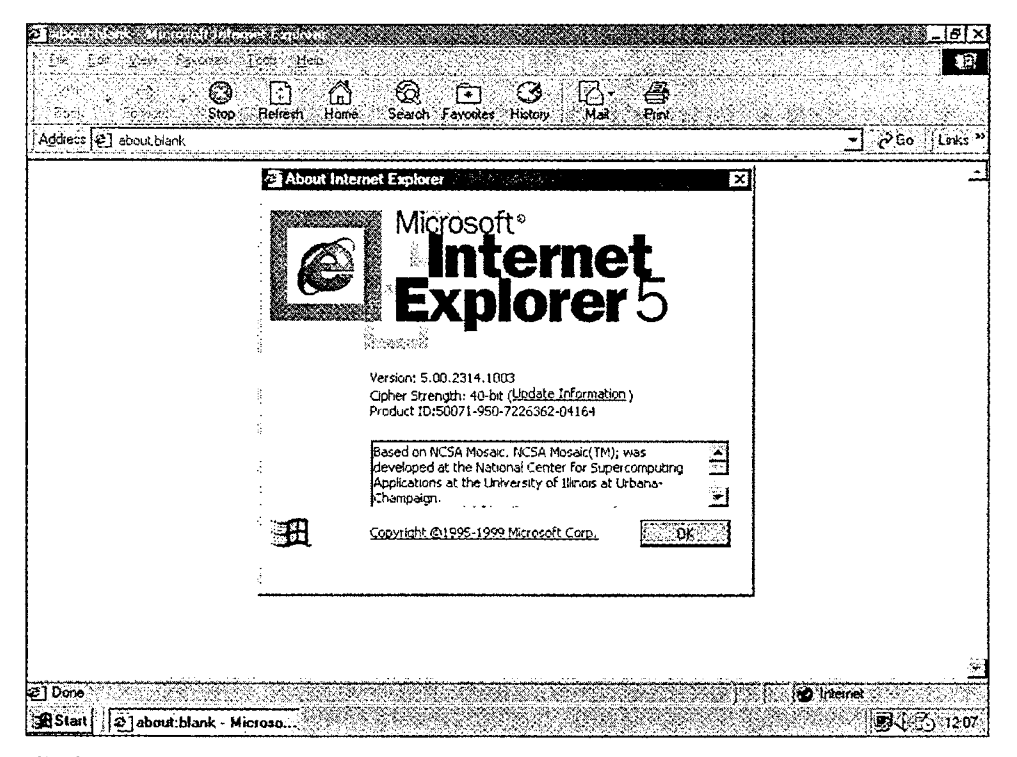
Требования, предъявляемые к программному обеспечению при установке SQL Server 2000

Итак, рассмотрим основные требования, предъявляемые к программному обеспечению при установке SQL Server 2000.

Прежде всего отметим, что единственной версией SQL Server 2000, для которой будут рассмотрены требования к необходимому программному обеспечению, является Personal Edition. Поскольку эта книга представляет собой всеобъемлющий источник информации по SQL Server 2000, здесь также будут упомянуты основные требования, которые предъявляются к операционным системам, поддерживающим SQL Server 2000.

Требования, предъявляемые к Windows 98

Операционная система Microsoft Windows 98 поддерживает установку SQL Server 2000 версий Personal Edition и Desktop Edition. Единственным требованием, предъявляемым к Windows 98, является наличие установленного броузера Internet Explorer 5.0 (как минимум). Microsoft выпустила несколько версий Windows 98, которые различались между собой в основном наличием в операционной системе пятой версии броузера Internet Explorer. Для того чтобы определить версию Internet Explorer, откройте меню Help (Справка) и выберите команду About Internet Explorer (О программе). Ниже показано, как должно выглядеть диалоговое окно, которое появляется в ответ на выполнение этой команды.



Информация о версии Internet Explorer

В том случае, если Internet Explorer 5.0 не установлен на вашем компьютере, его следует загрузить с Web-узла компании Microsoft. Копия Internet Explorer, предназначенная для загрузки, может быть найдена либо в каталоге загружаемого с данного Web-узла программного обеспечения, либо непосредственно по адресу `http://www.microsoft.com/ie/`.

Две основные причины, по которым на компьютере с операционной системой Windows 98 должен быть установлен Internet Explorer 5.0 (или выше), — это поддержка Microsoft Management Console (MMC) и способность просмотра в формате Hypertext Markup Language (HTML) справочных файлов, которые поставляются вместе с SQL Server 2000. При этом стоит отметить, что для наличия перечисленных свойств достаточно провести минимальную установку Internet Explorer (без установки каких-либо дополнительных модулей). Также следует упомянуть, что Internet Explorer не обязательно должен быть обозревателем, используемым вашей системой по умолчанию.

Требования, предъявляемые к Windows NT 4.0 Workstation и Windows 2000 Professional

В операционной системе Windows NT 4.0 Workstation должен быть установлен Internet Explorer 5.0 (как минимум) и пакет обновления Service Pack 4 (или более поздней версии). Эта операционная система (естественно, после установки SP 4 и IE 5) поддерживает установку SQL Server 2000 версий Personal Edition, Desktop Edition и Developer Edition.

Windows 2000 Professional поддерживает установку тех же версий SQL Server 2000, что и Windows NT 4.0.

Требования, предъявляемые к Windows NT 4.0 Server и Windows 2000 Server

Если у вас есть возможность установить SQL Server 2000 на серверную платформу, то мой вам совет: хватайтесь за нее двумя руками! Дело в том, что серверная установка SQL Server 2000 предполагает наибольшую гибкость в настройке и обслуживании этого программного продукта. При этом наиболее существенными преимуществами такого вида установки являются поддержка большего числа одновременно подключившихся пользователей и, как упоминалось ранее, возможность организации кластера.

Операционные системы Windows NT Server и Windows NT Server Enterprise Edition поддерживают установку SQL Server 2000 версий Enterprise Edition, Standard Edition, Personal Edition, Developer Edition и Desktop Edition.

Требованием, предъявляемым к операционной системе Windows NT 4.0 Server, является наличие установленного Internet Explorer 5.0 (как минимум) и Service Pack 4 (или более поздней версии).

На заметку Помимо установленного Internet Explorer 5.0, операционная система Windows NT 4.0 Server Enterprise Edition требует установки пакета обновления Service Pack 5.0 (или более поздней версии).

Операционные системы Windows 2000 DataCenter, Windows 2000 Advanced Server и Windows 2000 Server поддерживают установку тех же версий SQL Server 2000, что и Windows NT 4.0 Server. При этом стоит отметить, что, поскольку в момент написания этой книги Windows 2000 была самой последней версией операционных систем Microsoft, при ее использовании не требуется дополнительная установка программного обеспечения.

Требования, предъявляемые к Windows CE

Хотя это может показаться несколько странным, но до сих пор единственным и основным требованием, предъявляемым к операционной системе Windows CE, являлось ее наличие. Поскольку разработанная для CE версия SQL Server 2000 предназначена для сбора данных, подсоединения и последующей передачи этих данных версиям Enterprise Edition или Standard Edition, все предъявляемые к этой операционной системе требования были сведены к минимуму.

Следующие шаги

Если бы мы с вами работали вместе и я был бы вашим наставником, то я бы организовал процесс изучения основ SQL Server 2000 путем совместной разработки какого-нибудь приложения. Такой несколько необычный подход связан прежде всего с тем, что большинство людей лучше усваивают новый материал путем приобретения практического опыта, нежели путем получения теоретических знаний. Совместная разработка приложения не только поможет быстрее изучить основы SQL Server 2000, но и позволит разработать наиболее эффективный для вас способ изучения этого программного продукта.

Именно таким образом и построена эта книга: на протяжении всех ее глав описывается разработка приложения, по мере совершенствования которого изучаются все новые и новые аспекты разработки программ с использованием SQL Server 2000. Начиная с самой первой страницы, главный акцент делается не на синтаксисе используемых компьютерных языков, а на понимании рассматриваемой проблемы и поиске ее оптимального решения.

Некоторые наиболее пессимистично настроенные читатели могут спасовать перед таким поистине марафонским курсом обучения. Ха, попробовали бы они очутиться на моем месте!

В главе 1, "SQLSpyNet: от идеи до воплощения в виде базы данных SQL Server 2000", рассматривается коммерческая задача, ее предлагаемое решение и способ, благодаря которому SQL Server 2000 может использоваться для достижения этого решения. Помимо этого, описывается, как организовать соединение с только что установленным экземпляром SQL Server 2000 (для получения дополнительной информации обратитесь к приложению Б, "Установка и настройка SQL Server 2000").

Итак, можете откинуться на спинку стула, расслабиться и приготовиться к удивительному путешествию в мир баз данных.

Глава 1

SQLSpyNet: от идеи до воплощения в виде базы данных SQL Server 2000

В этой главе...

"Так какое же приложение мне придется создать "с нуля" с использованием для этого средств SQL Server 2000?" — этот вопрос наверняка волнует большинство читателей книги. Вместо того чтобы дать на него прямой ответ, в этой главе разъясняется основная идея создаваемого приложения и требования к фазе концептуального проектирования, которая обычно служит исходной точкой разработки любого приложения, взаимодействующего с базой данных.

Термин *Требования* (*requirements*) представляют собой некоторые зафиксированные характеристики приложения, согласованные между заказчиком и исполнителем. Приложение может считаться удачным, если оно, как минимум, удовлетворяет этим требованиям.

Познакомившись с заказчиком и с требованиями, предъявляемыми к приложению, попытаемся построить соответствующую ему модель данных, включающую в себя отношения между таблицами.

Изучив материал этой главы (кстати говоря, он является фундаментальным для всей последующей работы), читатель может приступить к установке и настройке SQL Server 2000, а также попробовать впервые подключиться к этой базе данных. Если же эта работа уже выполнена вами полностью или частично, проверьте все параметры: они должны соответствовать использующимся в нашем проекте. Настройка SQL Server 2000 подробно описывается в приложении Б, "Установка и настройка SQL Server 2000".

Изучение конкретного примера: Spy Net Limited

Представьте себе, что вы руководитель проекта Spy Net Limited (вымышленная компания, не имеющая какого-либо отношения к ныне здравствующим или уже умершим людям, а также к компаниям, которые либо уже зарегистрировались, либо еще нет).

Внешне Spy Net Ltd. ничем не отличается от любой рядовой организации, однако вместо того, чтобы покупать и продавать товары или заниматься чем-либо подоб-

ным, эта организация распределяет ресурсы между находящимися на задании тайными агентами.

В последнее время деятельность "плохих парней" достигла такого поистине угрожающего размаха, что распределение тайных агентов по различным заданиям и обеспечение их всеми необходимыми ресурсами крайне усложнилось. И вот, собрав в кулак все свое мужество, опираясь на опыт и блистательный ум, вы решили исследовать эту проблему и выяснить, доступно ли ее эффективное решение.

После проведения нескольких обширных (и, надо признаться, весьма специфических) исследований в поле вашего зрения попала поистине бесценная книга — и даже не просто книга, а исчерпывающее руководство по созданию базы данных тайных агентов что называется с нуля. (Все еще не замечаете совпадения?)

И хотя приведенный пример чисто гипотетический (если не сказать большего), затронутая нами проблема имеет самое прямое отношение к реальному положению вещей. Практически каждый день множество компаний встречаются лицом к лицу с одной и той же задачей (с которой встретились и наши вымышленные "заказчики"): огромное количество требующей учета информации и практически полное отсутствие ресурсов, необходимых для внедрения и управления системой учета.

Обзор приложения SQLSpyNet

SQLSpyNet — это вымышленная, но очень наглядная база данных, позволяющая получить знания, достаточные для изучения всего спектра возможностей SQL Server 2000. Кроме того, SQLSpyNet является *реляционной* базой данных (более подробно реляционные базы данных описаны ниже в главе), накапливающей информацию о тайных агентах и их заданиях, а также о злоумышленниках и их коварных планах, направленных на получение контроля над всей планетой.

SQLSpyNet наглядно описывает объекты реального мира и их взаимодействие друг с другом.

Термин Под *накоплением* (*capture*) подразумевается способность базы данных хранить введенную пользователем информацию. Аналогом накопления информации в базе данных является занесение очередного заказа покупателя в систему заказов гастрономического магазина.

Термин *Объект* (*object*) — одно из наиболее часто употребляемых понятий в мире компьютерных технологий. Наиболее общим определением этого термина является некая сущность, которую можно увидеть или до которой можно дотронуться (согласно такому определению человек — это объект). Объектами, например, могут быть части или какие-либо характеристики базы данных (причем это справедливо как для базы данных целиком, так и для ее отдельных таблиц), которые можно протестировать, изменить или к которым можно просто обратиться.

Приложение SQLSpyNet также предоставляет пользователю (каковым являетесь вы и ваша команда администраторов) возможность вводить и поддерживать критически важную для выполнения того или иного задания информацию, при необходимости создавая на ее основе подробный отчет. Примером такой критически важной информации может быть список тайных агентов, не находящихся в текущий момент ни на одном задании, а также список агентов, выполняющих одновременно слишком много поручений.

Естественно, доступ к просмотру и редактированию хранящейся в базе данных информации должен быть строго ограничен, что предполагает внедрение системы

безопасности в уже существующую модель приложения. Кто знает, быть может, от этого будет зависеть жизнь тайного агента!

Несмотря на строгие требования к безопасности, разрабатываемое приложение ни в коем случае не должно быть ориентировано только на одного пользователя. Приложение SQLSpyNet должно соответствовать трем базовым принципам: *масштабируемость*, *доступность* и *удобство сопровождения*.

Термин *Масштабируемость* (*scalability*) определяет критический предел дополнительной нагрузки, при которой приложение еще может выполняться без потери производительности. Другими словами, готово ли созданное приложение расширяться вместе с ростом объемов компании и увеличением количества пользователей, которые могут появиться в ближайшем будущем?

Термин *Доступность* (*availability*) — это возможность для пользователя получить доступ к хранящейся в базе данных информации, что в нашем случае аналогично возможности получить доступ к выполняющемуся приложению (это называется еще *периодом работоспособного состояния приложения*). В случае приложений, взаимодействующих с базами данных, требование к доступности часто задается схемой "24 часа в сутки, 7 дней в неделю" (или что-то максимально приближенное к этому).

Термин *Удобство сопровождения* (*maintainability*) тесно связно с последующим обслуживанием SQL Server 2000 и созданного на его основе приложения. В частности, оно может включать в себя степень простоты выполнения процедур резервного копирования и другие задачи администрирования.

Прежде чем приступить к разработке структуры приложения, следует подумать о типе информации, которая будет храниться в SQLSpyNet.

Определение основных функций приложения

Главной функцией нашего приложения является накопление информации о тайных агентах. Однако уже на этом этапе возникает несколько важных вопросов, касающихся прежде всего типа хранимой информации. Нужно ли хранить имена агентов? Их даты рождения? Каким образом тайный агент будет соотноситься с злоумышленником?

При разработке приложения будет создано несколько наиболее распространенных функций сбора данных, включая адрес, дату рождения и прозвище тайного агента или злоумышленника.

Предположим, что в данный момент в картотеке компании находится информация примерно о 20 тайных агентах и 10 самых опасных злоумышленниках планеты. Наша задача — организовать эту информацию таким образом, чтобы в любой момент к ней можно было быстро получить доступ и чтобы ее поддержка была связана с минимальными трудностями.

Создаваемое приложение должно иметь средства, с помощью которых можно было бы просмотреть отчет об активности тайных агентов и злоумышленников за определенный период. Подобная функциональная возможность поможет, например, выявить наименьшую и наибольшую активность "злых гениев" за год,

что позволит эффективно спланировать распределение агентов и ресурсов в определенное время года.

Подобно большинству других компаний, нам необходимо оставаться в рамках бюджета, что подразумевает строгий контроль за заработной платой агентов и различными накладными расходами.

Отслеживание и анализ финансовых потоков и трендов расширяет базовую модель нашего приложения до некоторого подобия системы управления. Имея подобную функциональную возможность, можно анализировать основные тенденции направления развития нашей компании. Когда разработка приложения будет находиться в стадии завершения, к нему можно будет добавить графические средства отображения линий трендов.

Следует отметить, что процессу анализа приложения уделяется в этой книге не слишком много внимания. Это связано прежде всего с тем, что ее главной темой является непосредственно SQL Server 2000, а не требования, предъявляемые к анализу разрабатываемого приложения. Тем не менее попытаемся придерживаться модели *Microsoft Solution Framework (MSF)*. Основная идея, которой необходимо руководствоваться при создании приложения, — это четкое понимание того, какая информация должна быть получена на выходе из базы данных в виде отчетов и какая информация может быть введена в базу данных.

Термин Модель *Microsoft Solution Framework (MSF)* — это созданный компанией Microsoft способ разработки и развертывания приложений. Концепция MSF слишком объемна для того, чтобы могла быть хотя бы частично рассмотрена в этой книге, однако ее основная идея — сосредоточиться на том, какую роль в процессе создания приложения играет каждый отдельный член команды разработчиков. Помимо этого, MSF предлагает набор руководящих правил, которые помогут завершить проект, не выходя за его временные и бюджетные рамки. Более подробную информацию о модели Microsoft Solution Framework можно найти на Web-узле компании Microsoft по адресу: `http://www.microsoft.com/msf/`.

Так что же нам нужно?

При разработке взаимодействующего с базой данных приложения SQLSpyNet преследуются такие цели:

- обеспечить возможность эффективного управления информацией о тайных агентах;
- реализовать механизм эффективного распределения ресурсов между различными заданиями;
- создать систему генерации отчетов о текущей деятельности тайных агентов;
- помешать осуществлению задуманных злоумышленниками планов и таким образом сберечь мир на планете.

Итак, мы пришли к четкой формулировке цели, которой необходимо достичь: создать масштабируемое, безопасное и надежное приложение, взаимодействующее с базой данных, которое бы намного облегчило ввод, обработку и извлечение секретной информации.

Экскурс

Измерение "успеха"

Наиболее важным моментом, который необходимо учитывать при разработке любого приложения, является четкая формулировка и доступность поставленных целей не только для коллектива разработчиков, но, что очень важно, для заказчиков проекта. Постарайтесь выразить в каких-либо количественных показателях все результаты, достигнутые при создании приложения, так как в противном случае у заказчиков будет шанс усомниться в его полезности.

В следующих разделах этой главы рассматриваются способы достижения поставленной цели.

Моделирование приложения SQLSpyNet

Как уже упоминалось в предыдущих разделах, основной целью является создание приложения, позволяющего вести учет находящихся в нашем распоряжении тайных агентов и отслеживать бурную деятельность злоумышленников.

Некоторые из основных требований, предъявляемых к создаваемому приложению, были сформулированы в предыдущем разделе. При разработке системы вы должны постоянно держать эти требования в уме, но не для того, чтобы систематически сверять с ними каждую строку программного кода, а для того, чтобы не допустить ситуации, когда одна из поставленных целей достигается за счет пренебрежения другой целью. После окончания работ над определенной частью (модулем) системы следует оценить, соответствует ли эта система основным требованиям.

Экскурс

Создайте собственную систему измерений

Требования к приложению, согласованные между заказчиком и исполнителем, должны быть *измеримы* (*measurable*). Как упоминалось ранее, это поможет провести более эффективную и, что самое главное, объективную оценку проделанной работы. Кроме того, четкая оценка приложения способствует извлечению правильных выводов из проекта, что, несомненно, позволит разработчику более ответственно и профессионально отнестись к *оценке* следующего приложения.

Термин

Оценка (*scoping*) — это процесс определения границ проекта. Оценивая проект, необходимо согласовать с заказчиком все предъявляемые требования и, что самое главное, прийти к единому мнению относительно целей, которые должны быть достигнуты в рамках отведенного отрезка времени.

Проектирование таблиц приложения SQLSpyNet

Разобравшись с предъявляемыми к приложению функциональными требованиями, необходимо рассмотреть вопрос соответствия объектов реального мира их виртуальным аналогам, выяснить связи, которые существуют между этими виртуальными объектами (*сущностями*), а также определить те подробности (*свойства*), с помощью которых будет охарактеризован каждый человек или объект.

Термин

Сущность (*entity*) — это некая модель объекта реального мира (как правило, сущность представляется таблицей базы данных). Примером сущности может быть человек.

Термин Каждому человеку ставятся в соответствие *свойства* (*properties*), или *атрибуты* (*attributes*), которые могут включать в себя имя, общественное положение, адрес проживания, а также то (самое главное!), на чьей стороне находится этот человек.

Суммируя сказанное, мы приходим к необходимости следующей детализации используемых сущностей:

- "тайные агенты" — имя, адрес проживания (включая регион планеты), общественное положение и заработная плата;
- "плохие парни" — имя, адрес проживания (включая регион планеты), общественное положение и прозвище;
- "деятельность" — тип деятельности, задействованные тайные агенты и злоумышленники, а также исход этой деятельности (кто победил и когда).

Определившись с объектами реального мира и поставив перед собой конкретные цели, можно приступать к разработке *диаграммы отношений между объектами*, или, как ее еще называют, *диаграммы "сущность–связь"* (*Entity Relationship Diagram — ERD*) приложения. Разумеется, нам потребуется хранить информацию о тайных агентах, которую логичнее всего было бы представить соответствующей таблицей. Это же относится и к информации о злоумышленниках. Процесс определения структур данных, необходимых для хранения информации приложения, называется *моделированием данных*.

Термин *Моделирование данных* (*data modeling*) — это процесс наглядного изображения схемы работы приложения, потоков данных, а также способа связи отдельных таблиц и элементов приложения. Моделирование данных осуществляется с применением различных средств, включая *диаграмму потоков данных* (*Data-Flow Diagram — DFD*), *диаграмму состояний* (*State-Transition Diagram — STD*) и *диаграмму "сущность–связь"* (*Entity Relationship Diagram — ERD*).

Термин *Диаграмма "сущность–связь"* является одной из ключевых концепций, применяющихся в процессе разработки баз данных. С помощью этой диаграммы приложение может быть тщательно спроектировано (на бумаге или с применением какой-либо специализированной программы, например EasyCase, которая использовалась при написании этой книги), прежде чем его модель будет реализована в виде базы данных. Диаграмма отношений между объектами включает в себя логические (виртуальные) объекты, которые были использованы для представления в приложении объектов реального мира, и взаимоотношения между этими логическими объектами. В нашем примере диаграмма "сущность–связь" определяет взаимоотношения между тайными агентами и злоумышленниками.

Разрабатывая диаграмму "сущность–связь", важно не увязнуть во множестве связей, необходимых полей таблиц и прочих мелочах. Самая главная часть этого процесса — четкое выражение идеи, положенной в основу проекта приложения. Как-никак ничто не вечно, а потому при необходимости мы всегда сможем вернуться к пересмотру проекта приложения.

Использование реляционной теории для моделирования SQLSpyNet

Базовая реляционная теория достаточно проста, хотя понимание этого приходит только после ее скрупулезного изучения. Реляционная теория предназначена для описания определенных отношений, сложившихся между двумя сущностями. В некоторых приложениях эти отношения могут принимать довольно сложную форму, однако если проанализировать приложение, разложив его на составные части, то по способу определения связей между сущностями можно легко выявить логику работы такого приложения.

Экскурс

Теория и революция

Как уже упоминалось в предыдущих разделах, база данных строится на сущностях и их свойствах, которые принимают соответственно форму строк и столбцов таблицы.

Необходимо знать, что в настоящей реляционной теории для описания каждой из этих концепций используются несколько иные термины, да и вообще вся теория носит более "академический" характер. Однако вряд ли вы купили эту книгу для прохождения университетского курса, так что впоследствии будут употребляться только те термины и понятия, которые необходимы для разработки приложения SQLSpyNet. Несмотря на это, стоит порекомендовать всем, кто хочет углубиться в теорию построения баз данных (а для администраторов баз данных это обязательная норма), непременно изучить как можно больше аспектов реляционной теории.

Итак, реляционная теория служит для определения *отношений* между сущностями, но и это, к сожалению, еще не объясняет ее главного предназначения. На самом деле реляционная теория всего лишь описывает способ хранения и просмотра данных.

Реляционная теория служит для задания структуры данных и определения взаимоотношений между похожими наборами данных. Во многом она базируется на теории множеств, которая преподается либо еще в школе, либо на первых курсах университета. (Никогда бы не подумал, что мне придется вспоминать ту чепуху, которую мы учили в школе!)

Термин

Отношение (*relationship*) между сущностями — это виртуальная связь, как правило основанная на взаимоотношении объектов реального мира. Например, поскольку каждый тайный агент и злоумышленник являются людьми, между ними можно создать некоторую виртуальную связь (отношение). Отношения принимают одну из трех форм: "один к одному", "один ко многим" и "многие ко многим". Далее в этом разделе рассматривается способ, с помощью которого нужный тип отношения может быть определен на основе анализа взаимоотношения объектов реального мира.

Наверняка одним из самых лучших способов описания реляционной теории является ее практическое применение на примере разрабатываемого приложения. Сначала рассмотрим сущности и их атрибуты, которые должны присутствовать в проектируемой базе данных, а затем попробуем описать существующие между ними связи.

Тайный агент — это человек (объект), имеющий определенные характеристики, которые могут быть представлены в виде атрибутов соответствующей сущности: фамилия, имя, дата рождения и т.д. Примерная модель объекта "тайный агент" (Spy) изображена на рис. 1.1.

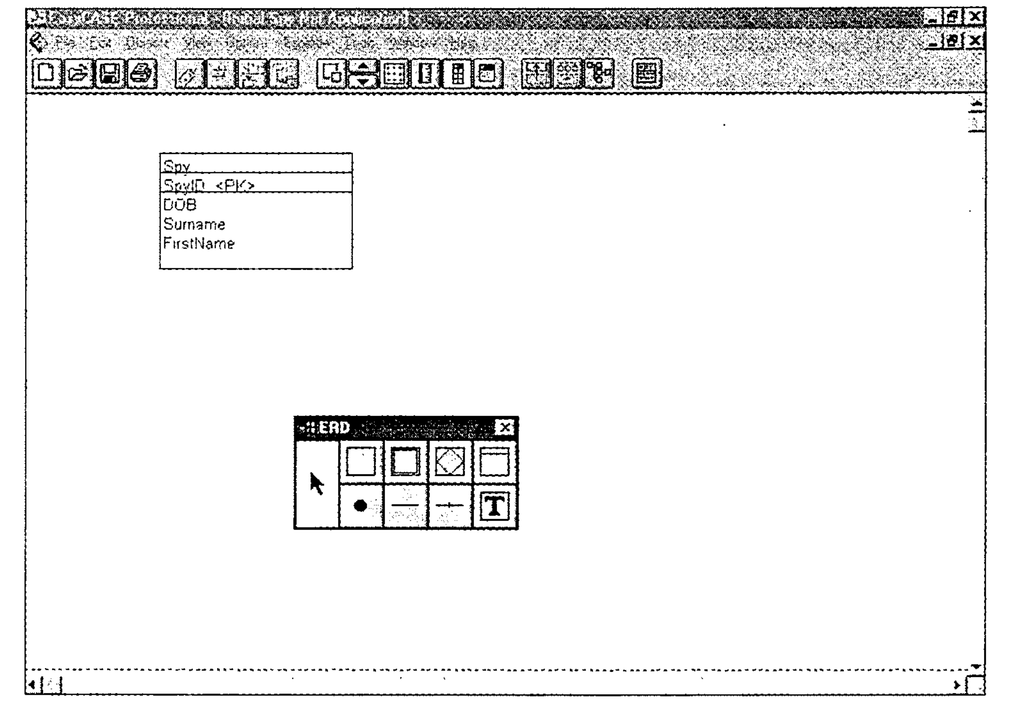


Рис. 1.1. Модель объекта "тайный агент"

Кто же еще задействован в нашем приложении? Естественно, это так называемый злоумышленник, чья модель будет очень похожа на модель тайного агента (она имеет практически аналогичный набор атрибутов). Примерная модель объекта "плохой парень" (BadGuy) изображена на рис. 1.2.

Следующим шагом является определение отношения между объектами "тайный агент" и "плохой парень". Одной из самых явных связей является тот факт, что тайный агент противостоит злоумышленнику и наоборот. Один тайный агент может сражаться сразу против целой армии злоумышленников (одновременно или на протяжении некоторого времени), это же утверждение имеет силу и для злоумышленника. Он может противостоять одновременно или на протяжении некоторого отрезка времени нескольким или одному тайному агенту.

Экскурс

Обязательно выберите тип отношений между двумя сущностями

Для того чтобы безошибочно определить тип отношения между двумя сущностями, необходимо ответить на несколько наводящих вопросов: "Может ли одна строка в таблице A ("тайные агенты") соответствовать более чем одной строке в таблице B ("плохие парни")? И наоборот, может ли одна строка в таблице B соответствовать более чем одной строке в таблице A?".

Если вы ответили отрицательно на оба вопроса, то отношение между двумя сущностями имеет тип "один к одному" (обозначается как 1-1).

Если на первый вопрос вы ответили "нет", а на второй — "да", то тип отношения между двумя сущностями — "один ко многим" (1-N).

И наконец, если вы ответили "да" на оба вопроса, то отношение между двумя сущностями имеет тип "многие ко многим" (N-M).

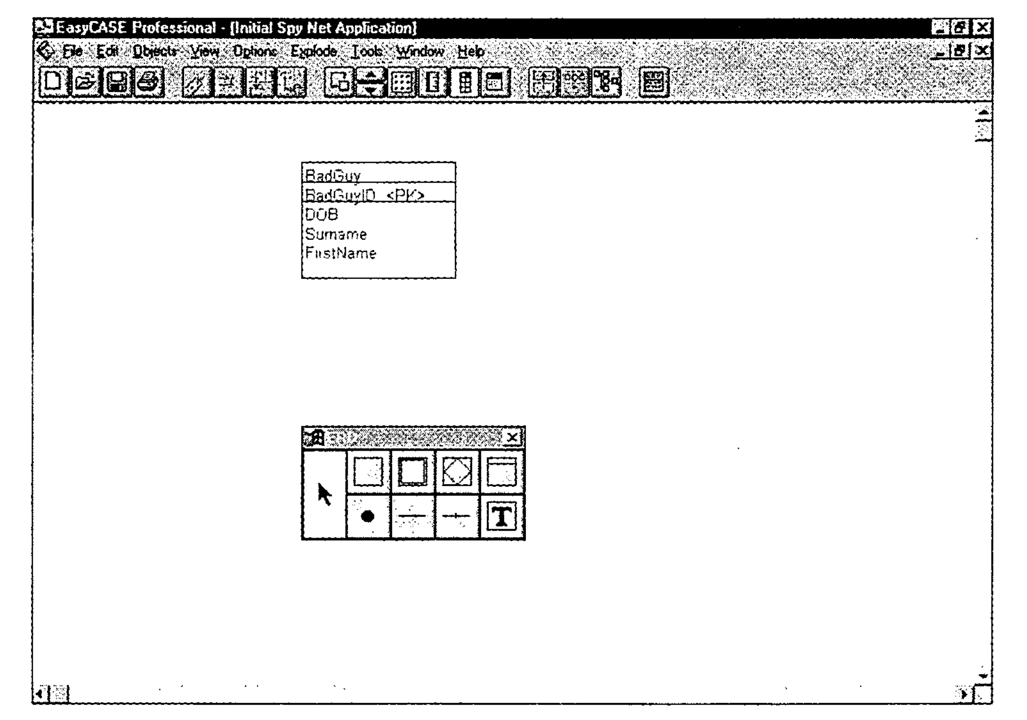


Рис. 1.2. Модель объекта "плохой парень"

Руководствуясь ответами на приведенные выше вопросы, получим, что тип отношения между сущностями "тайный агент" и "плохой парень" — "многие ко многим".

На рис. 1.3 изображено наглядное представление отношения типа "многие ко многим".

Представление объектов в виде таблиц базы данных

Присмотревшись к структуре представленных на диаграмме таблиц, можно задать себе вопрос: а не являются ли они фактически представлением одной и той же сущности? Ведь и тайные агенты, и злоумышленники — это люди, которые к тому же имеют достаточно много общих характеристик (имя, дата рождения, адрес проживания и прозвище). Единственной существенной разницей между ними является та сторона, на которую они работают (или которую они представляют). Учитывая сказанное, не будет ли проще объединить эти сущности в одну общую таблицу, внеся при этом только незначительные изменения в атрибуты?

Экскурс

Процесс разработки должен приносить удовольствие

Проектирование структуры базы данных — процесс субъективный. Вполне корректная модель, с точки зрения одного проектировщика, может оказаться совсем непригодной, с точки зрения другого. Возвращаясь к нашему примеру, попробуем найти ответ на один весьма спорный вопрос: стоит ли представлять объекты "тайный агент" и "плохой парень" одной или двумя разными таблицами? Главным преимуществом при объединении этих двух сущностей в одну таблицу является возможность обновления соответствующей им информации в одном и том

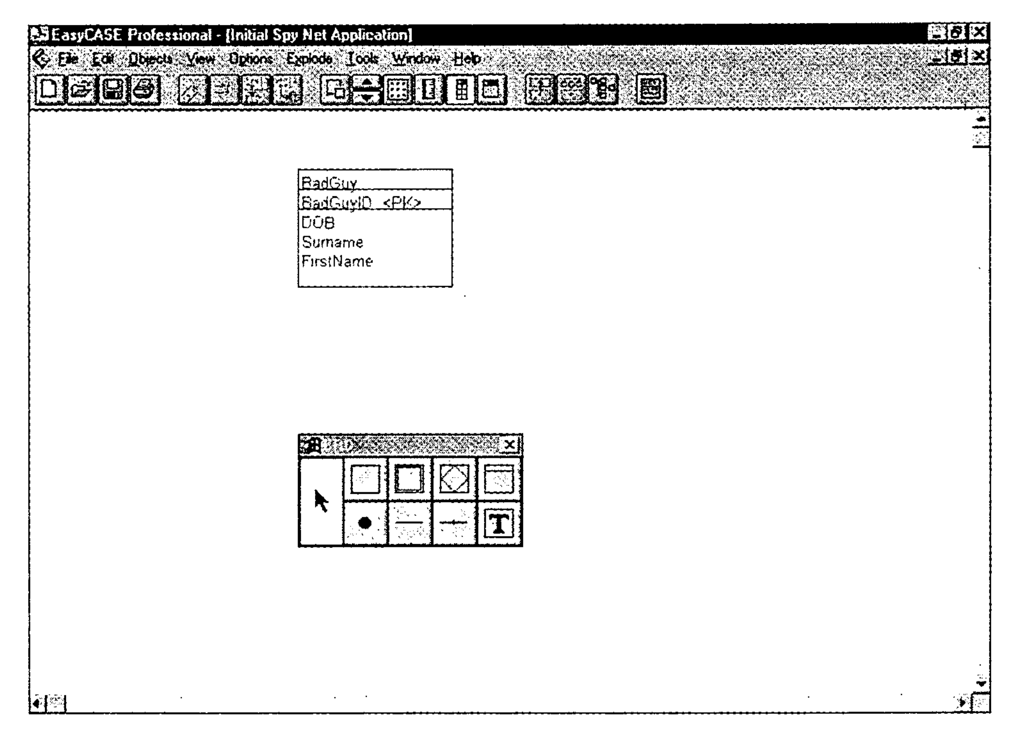


Рис. 1.3. Графическое представление отношения типа "многие ко многим"

же месте, недостаток такого подхода состоит в необходимости использовать дополнительные средства для отличия тайных агентов от злоумышленников.

Экскурс

Разумеется, при написании этой книги соответствующее решение было принято, так что читателю пока отводится всего лишь скромная роль статиста. Когда вы будете готовы к самостоятельному созданию базы данных, еще раз обратите внимание на все аспекты, рассмотренные в этой главе, а также в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", и поразмыслите над другими вариантами проектирования базы данных. Окончив чтение книги, вы можете еще раз создать приложение SQLSpyNet, используя при этом другую методологию проектирования базы данных, но преследуя те же цели и имея в качестве исходных данных ту же самую информацию, а затем сравнить полученный результат с тем, который будет достигнут в конце данной книги.

Однако, как подсказывает опыт, наиболее легкий способ получить знания — извлечь их из других проектов. Иногда 10–20 строк программного кода могут дать понимание концепций, на описание которых в отдельных книгах отводятся целые главы. Попробуйте рассмотреть программный код некоторого приложения (в этом смысле служба MSDN компании Microsoft — поистине бесценный источник) или дизайн базы данных, и вы поймете, что изучение результатов труда других разработчиков может сослужить хорошую службу в смысле получения определенных знаний. Как правило, большинство пользователей вовсе не против прервать свои занятия для того, чтобы объяснить суть некоторой идеи или концепции. Единственное правило, которое при этом следует усвоить, гласит: никогда не бойся задать лишний вопрос!

Множество концепций реляционных баз данных находится практически у вас под рукой. Устанавливая на компьютер копию SQL Server 2000, при желании

Экскурс можно установить также и несколько демонстрационных баз данных (*Pubs* и *Northwind*). Они представляют собой прекрасный пример полнофункциональных реляционных баз данных, предоставляя не только великолепный материал для изучения, но и шанс поэкспериментировать с реальной базой данных.

Итак, не бойтесь спрашивать (даже если вопросы кажутся вам бессмысленными), ищите доступные ресурсы и, что самое главное, получайте от жизни удовольствие!

Объединение похожих таблиц в одну снизит уровень сложности приложения и сделает структуру базы данных более "плоской" (этот процесс также называется *денормализацией* базы данных). К сожалению, подобная практика может привести к появлению нежелательных побочных эффектов. В главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", рассматривается влияние нормализации на процесс проектирования структуры базы данных и вытекающие из этого последствия.

На данный момент вам следует знать только то, что исповедуемая в книге философия разработки структуры базы данных предполагает сведение к абсолютному минимуму одного из самых распространенных побочных эффектов — появления значений `null` в полях базы данных (количество значений `null` прямым образом влияет на так называемую *избыточность информации*).

Термин `Null` — это специальное значение, которое используется в реляционной теории и теории баз данных. Парадоксально, но оно указывает на отсутствие какого-либо значения в поле таблицы. Значение `null` не равняется пустой строке ("") или нулю (0). Таким образом, операции сравнения "больше, чем" (>), "меньше, чем" (<) и операция проверки равенства (=) не могут быть применены для сравнения этого значения даже с другим таким же значением `null`.

Термин *Избыточность информации* (*redundant data*) — это повторение определенных данных более чем один раз в приложении. Хранить одну и ту же информацию в нескольких различных местах весьма неэффективно, так как это ведет к усложнению поддержки такой базы данных и увеличению расходов на устройства хранения.

Использование подтипов для повышения гибкости SQLSpyNet

Для того чтобы облегчить задачу сведения к минимуму значений `null`, следует воспользоваться так называемыми *подтипами*. Подтипы не только помогают спроектировать структуру приложения с минимальным числом значений `null`, но и обеспечивают значительную гибкость при будущем расширении проекта.

Термин Для тех, кто знаком с объектно-ориентированным программированием, концепции выделения подтипов и наследования не являются чем-то новым. По существу, *подтип* (*subtype*) позволяет взять свойства (атрибуты) заданного элемента (например, таблицы) и воспроизвести новый элемент, который будет иметь те же самые атрибуты. Этот процесс называется наследованием. Важно отметить, что получившийся в результате наследования новый элемент может иметь атрибуты, отличные от унаследованных.

Для того чтобы добавить в уже развернутое приложение новый объект, следует всего лишь создать еще один подтип ("наследник") заданного супертипа. Имея, на-

пример, супертип "человек", можно выделить из него подтип "служащий", который будет характеризоваться всеми теми же атрибутами, что и супертип, и обладать при этом некоторыми уникальными для этого подтипа свойствами, например свойством "заработная плата". Напомним, что подтип наследует все свойства супертипа, в данном случае одним из них будет свойство "имя".

Термин *Супертип* (*super-type*) — это "предок" по отношению к подтипу (см. определение выше). Подтип наследует все атрибуты соответствующего ему супертипа. Например, "человек" является супертипом по отношению к подтипу "тайный агент", поскольку "тайный агент" имеет некоторые атрибуты, унаследованные из сущности "человек".

Внесение изменений в структуру базы данных требует внесения изменений и в диаграмму отношений между объектами, которая теперь вместо двух сущностей с одинаковыми атрибутами будет иметь три, причем сущности "тайный агент" и "плохой парень" наследуются от "человек". Получившаяся в результате этого диаграмма показана на рис. 1.4.

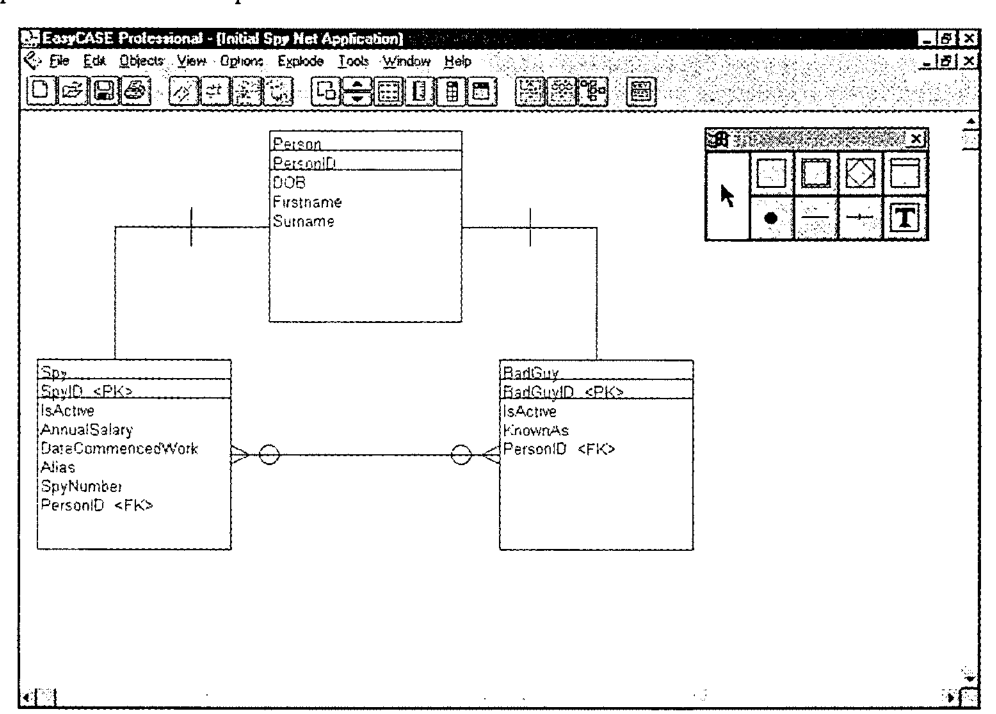


Рис. 1.4. Новая диаграмма отношений между объектами, учитывающая примененную к структуре базы данных концепцию выделения подтипов

Легко видеть, что усовершенствованная модель структуры базы данных позволяет без труда достичь по крайней мере одного из принятых выше требований к разрабатываемому приложению. Речь идет о возможности масштабирования проекта без внесения большого количества изменений в программный код. Теперь для того, чтобы внести в структуру базы данных еще один тип людей, следует всего лишь выделить подтип супертипа "человек" и задать его отличительные атрибуты. Вместе с тем введение супертипа позволяет уменьшить ряд проблем, с которыми мы сталкивались на ранних этапах проектирования приложения.

Определение взаимоотношений между заклятыми врагами

Как упоминалось в предыдущем разделе, между объектом "человек" и объектами "тайный агент"/"плохой парень" существует естественное взаимоотношение. Программным эквивалентом *отношения* между супертипом и подтипом является один из способов установки связи между объектами данных для обеспечения *целостности на уровне ссылок*.

Чтобы определить правила, обеспечивающие целостность на уровне ссылок и устанавливающие отношение между логически связанной информацией, следует воспользоваться такими средствами базы данных, как *первичный ключ* и *внешний ключ*.

Термин Атрибут, однозначно определяющий строки таблицы, называется *первичным ключом* (*primary key*). Примером первичного ключа может служить идентификационный код налоговой инспекции. Поскольку каждый человек имеет собственный идентификационный код, его можно использовать для однозначного определения личности.

Термин Если к одной таблице добавить атрибут, который является первичным ключом другой таблицы, то добавленный атрибут будет называться *внешним ключом* (*foreign key*). Примером внешнего ключа может служить все тот же идентификационный код налоговой инспекции (являясь первичным ключом таблицы "служащие", в этом случае он выступает уже как внешний ключ), который используется работодателем для связи таблицы служащих и таблицы налоговых сборов. Посредством идентификационного кода связываются таблицы "служащие" и "заработная плата".

Термин *Целостность на уровне ссылок* (*referential integrity*) — это процесс поддержки соответствия между логически связанными таблицами. В соответствии с принципом целостности на уровне ссылок невозможно внести запись в таблицу "заработная плата" прежде, чем соответствующая запись появится в таблице "служащие".

Собрав необходимые сведения для приложения SQLSpyNet или любой другой базы данных, следует тщательно проанализировать все естественные взаимосвязи, существующие внутри исходной информации. Было бы опрометчиво организовать касающиеся тайных агентов сведения, включая идентификационный номер, имя, адрес, заработную плату, общественное положение и т.д., в одну громадную таблицу. Подумайте, что будет в том случае, если вам придется изменить ставку налога для каждого тайного агента, общее число которых будет уже не 20, а 20 тысяч? Самым простым выходом из подобного положения будет *запрос*, который должен обработать 20 тыс. строк таблицы. В то же время после тщательного анализа исходной информации, размещения ее в нескольких таблицах и установки между ними отношения с использованием первичных и внешних ключей решением будет одно-единственное изменение данных.

Термин *Запросы* (*queries*) предназначены для изменения, удаления и извлечения информации из базы данных. При построении запроса используются специальные команды. Главное преимущество запросов заключается в том, что они могут создаваться динамически или быть заранее предопределенными. Более подробно запросы рассматриваются в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet".

Каждый элемент реляционной базы данных должен быть однозначно определяем. И в соответствии с этим имена таблиц должны быть уникальны. Запрещается, например, иметь две таблицы с именем Spy или вводить в таблицу две совершенно одинаковые строки.

Для обеспечения однозначного определения строк таблицы используется первичный ключ. Он представляет собой столбец (атрибут) таблицы, данные которого уникальны. В приведенной на рис. 1.5 таблице таким столбцом будет первый (`SocialSecurityNo`). Таким образом, становится очевидно, что именно этот столбец и должен быть объявлен первичным ключом.

SocialSecurityNo	Surname	FirstName	DOB
123456	Bloggs	Joe	1/12/72
654321	Hendrix	James	1/12/72
742356	Hawthorne	Rob	1/12/72
564231	Hawthorne	Mike	1/12/72
654124	Happy	Joe	1/12/72
124561	Happy	Smile	1/12/72

Рис. 1.5. Определение первичного ключа таблицы

С первичными ключами все ясно, однако остается открытым еще один очень важный вопрос: каким образом реализовать отношение между двумя логически связанными таблицами? Здесь на помощь приходит так называемый внешний ключ. Внешний ключ данной таблицы представляет собой столбец, являющийся первичным ключом другой таблицы, с которой необходимо установить отношение. На рис. 1.6 изображены таблицы А и В, каждая из которых имеет первичный ключ (отмеченный изображением маленького ключика).

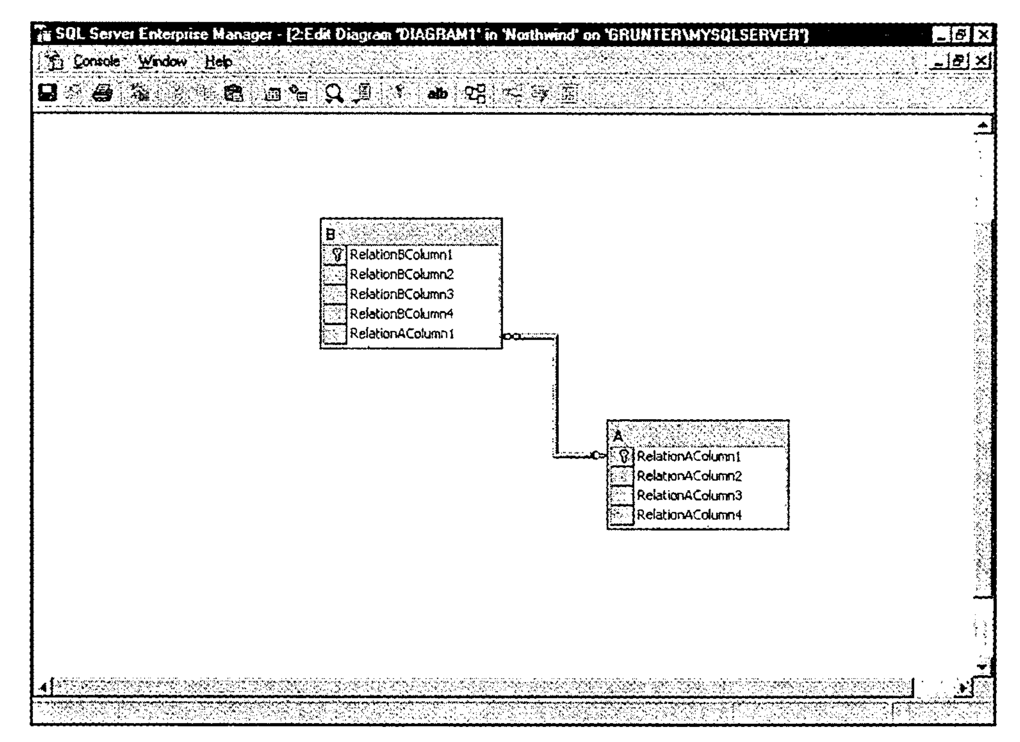


Рис. 1.6. Графическое представление отношения между таблицами

Обратите внимание на то, что в таблице B содержится столбец таблицы A, являющийся ее первичным ключом. Таким образом, можно утверждать, что таблица B связана (имеет отношение) с таблицей A посредством внешнего ключа.

Размещение первичного ключа таблицы A в таблице B позволяет создать отношение типа "один ко многим". Другими словами, одной строке таблицы A может соответствовать несколько строк таблицы B. На рис. 1.6 отношение такого типа обозначается линией, соединяющей внешний ключ таблицы B с первичным ключом таблицы A, причем обозначенный изображением маленького ключика конец этой линии указывает на первичный ключ.

На заметку Для того чтобы указать сторону "многие" отношения типа "один ко многим", в SQL Server 2000 используется изображение символа бесконечности (∞), расположенное на конце линии, обозначающей отношение между таблицами.

Уточнение отношения между таблицами *Spy* и *BadGuy*

Рассматривая в одном из предыдущих разделов этой главы диаграмму отношений между объектами, мы объявили тип отношения между таблицами "тайные агенты" и "плохие парни" как "многие ко многим". И хотя такой тип отношения, несомненно, корректен на этапе начального моделирования системы, он не может быть реализован в чистом виде с помощью реляционной базы данных.

Удивлены? Думаю, да. Как правило, сталкиваясь впервые с реляционной теорией и проектированием баз данных, очень трудно осознать необходимость введения такого типа отношения между таблицами, который не может быть в чистом виде реализован на практике.

Суть этого парадокса состоит в том, что реляционная теория имеет под собой чисто математическую основу. Отношение типа "многие ко многим" оказывается весьма полезным на этапе проектирования приложения, поскольку дает возможность абстрагироваться от деталей реализации и сфокусировать внимание на общей картине разрабатываемой системы. Уточнение модели с учетом ограничений, накладываемых в результате использования конкретной технологии (именно этому и посвящается данный раздел), обычно осуществляется на последующих этапах разработки.

Причина, по которой отношение типа "многие ко многим" не может быть реализовано в чистом виде, заключается в том, что заранее не известно, какое количество строк одной и другой таблицы будет определять связь между ними. Если все-таки попытаться реализовать модель отношения между таблицами такого типа, то это неизбежно приведет к появлению повторяющихся строк, что нарушает правило *нормализации*.

Несмотря на то что используемая база данных (впрочем, как и все реляционные базы данных) не поддерживает отношение типа "многие ко многим", необходимо отыскать способ реализации такого отношения с помощью доступных средств. Для того чтобы реализовать это отношение с использованием реляционной базы данных, следует познакомиться с понятием *ассоциативной* (или *объединяющей*) таблицы.

Термин *Ассоциативная таблица* (*associative table*) создается только для того, чтобы реализовать отношение типа "многие ко многим", которое может существовать между двумя таблицами. В результате использования ассоциативной таблицы одно отношение типа "многие ко многим" заменяется двумя отношениями типа "один ко многим", что достигается включением в ассоциативную таблицу первичных ключей связанных таблиц (первичные ключи этих таблиц становятся внешними ключами ассоциативной таблицы).

Таким образом, ассоциативная таблица полностью оправдывает свое название, поскольку она, по существу, используется для создания отношения (ассоциации) между двумя таблицами. В результате использования ассоциативной таблицы одно сложное отношение типа "многие ко многим" разбивается на два отношения типа "один ко многим".

Возвращаясь к таблицам "тайные агенты" и "плохие парни", введем новую ассоциативную таблицу, которая будет иметь название *Activity* (Деятельность). Эта таблица хранит информацию о встречах между тайными агентами и злоумышленниками и фиксирует результаты этих встреч.

На рис. 1.7 изображена новая диаграмма отношений между объектами, которая включает в себя ассоциативную таблицу.

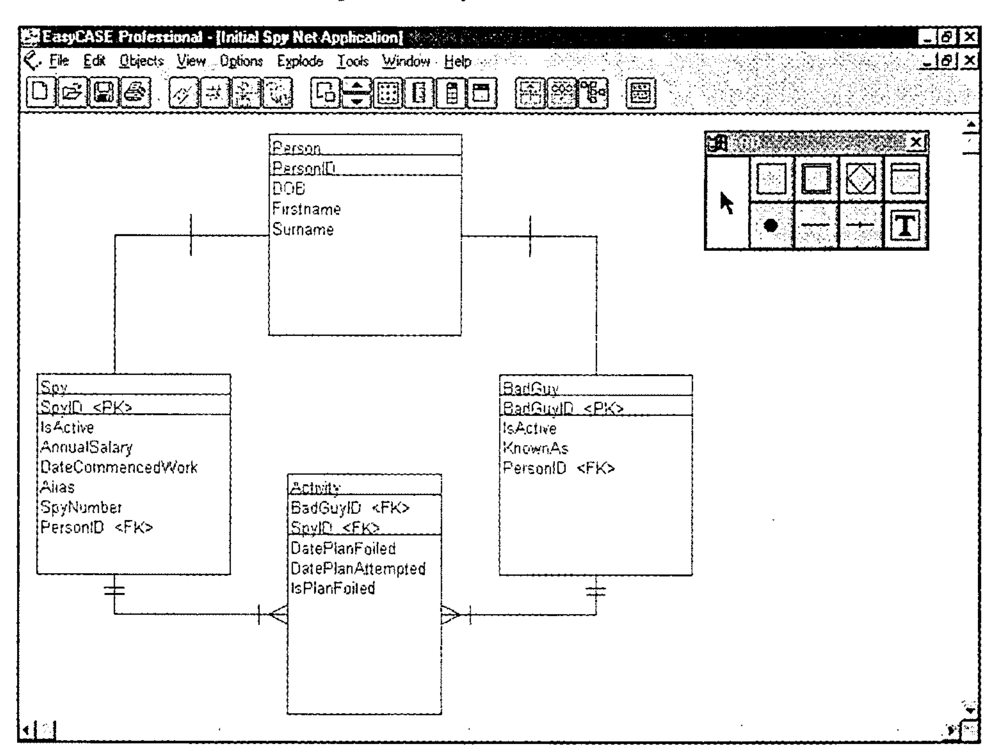


Рис. 1.7. Использование ассоциативной таблицы позволяет заменить отношение типа "многие ко многим" двумя отношениями типа "один ко многим"

На заметку Обратите внимание на первичный ключ таблицы "деятельность". Ключ такого типа называется *сцепленным* (concatenated) первичным ключом. Он состоит из двух объединенных внешних ключей других таблиц.

На некоторое время нам придется отложить изучение реляционной теории и проектирование нашего приложения. Разработанная на этот момент модель, безусловно, не является совершенной, однако вы уже имеете достаточный запас знаний для начала работы над проектом. В главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", мы вернемся к вопросам разработки структуры приложения и попробуем оценить ее сложность.

Обзор приложения SQLSpyNet

SQLSpyNet — это та идея, которая объединяет в себе все рассматриваемые в данной книге технологии. Без приложения, разработка которого описывается в книге, последняя бы превратилась просто в набор примеров.

SQLSpyNet собирает в единое целое те функциональные "кусочки", которые создаются на протяжении более чем десяти глав книги, давая возможность читателю ощутить себя полноценным разработчиком приложения, использующего SQL Server 2000.

К концу книги вы будете иметь не только полностью функционирующее приложение (удовлетворяющее при этом всем требованиям заказчика), но еще и мощную библиотеку программного кода, который может пригодиться в будущем при разработке подобных приложений. Более того, одной из особенностей приложения SQLSpyNet является Web-ориентированный интерфейс, создание которого описывается в главе 12, "Разработка интерфейса пользователя базы данных SQLSpyNet". Таким образом, вместе с навыками разработчика баз данных вы получите еще и некоторые знания, касающиеся создания Web-узлов!

Приложение SQLSpyNet может быть использовано даже в качестве некоторого подобия стратегической игры! Игроки могут задать тайным агентам и злоумышленникам определенные поручения и вести счет в зависимости от успехов своей стороны.

Схема разрабатываемой базы данных может быть адаптирована для накопления информации о служащих и о работах, на которых они в данный момент заняты. Таким образом, приложение превращается в систему учета рабочего стажа.

В принципе вы даже можете создать собственную организацию тайных агентов и использовать уже готовое приложение в качестве образца построения ее структуры. Вне зависимости от выбора, к концу книги получится приложение, которым можно будет гордиться. Ведь на его создание было потрачено столько времени и усилий!

В оставшейся части книги рассматриваются вопросы непосредственной разработки и внедрения нашего приложения. Для этого прежде всего необходимо создать соединение с SQL Server 2000. Если SQL Server 2000 все еще не установлен или установлен, но к нему не организовано подключение, обратитесь к приложению Б, "Установка и настройка SQL Server 2000", и выполните все указанные там действия.



Некоторые из наиболее активных читателей наверняка давно уже установили на свой компьютер SQL Server 2000. Самые прогрессивные из них, видимо, уже успели даже создать соединение с этой базой данных. Тем не менее следует еще раз напомнить, что для корректной работы приложения SQLSpyNet параметры установки SQL Server 2000 и подключения играют крайне важную роль, особенно если ваш компьютер работает под управлением операционных систем Windows 2000 или Windows NT. За более подробной информацией о параметрах установки SQL Server 2000 и подключения обратитесь к приложению Б, "Установка и настройка SQL Server 2000"..

Резюме

Итак, мы определились с основными требованиями, предъявляемыми к разрабатываемому приложению со стороны заказчика, и с ограничениями, накладываемыми в соответствии с этими требованиями на проект приложения. Следующим эта-

пом будет путешествие в мир SQL Server 2000, проводником по которому является средство Enterprise Manager (прежде чем начать это путешествие, следует установить соединение с SQL Server 2000, как описывается в приложении Б, "Установка и настройка SQL Server 2000"). Единственное напутствие, которое необходимо дать перед "путешествием", таково: не сохраняйте никаких изменений до тех пор, пока не будете полностью уверены в том, что делаете!

Следующие шаги

В главе 2, "Компоненты SQL Server 2000", рассматриваются основные средства разработки и управления, которые поставляются вместе с SQL Server 2000. В процессе установки и настройки SQL Server 2000 в соответствии с указаниями, приведенными в приложении Б, вы должны были познакомиться по крайней мере с одним таким средством — Enterprise Manager. Помимо Enterprise Manager, в главе 2 рассматриваются остальные средства SQL Server 2000, которые так или иначе используются на протяжении этой книги.

В главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", мы переведем наших тайных агентов и злоумышленников из "бумажного" формата в формат объектов SQL Server 2000 с добавлением некоторых дополнительных настроек и фактическим началом разработки базы данных.

Итак, соберитесь с силами и приготовьтесь окунуться в захватывающий мир проектирования, разработки и развертывания приложения в SQL Server 2000.

ГЛАВА 2

Компоненты SQL Server 2000

В этой главе...

SQL Server 2000 — мощная и многофункциональная система управления базами данных (СУБД). Она имеет дружественный и интуитивно понятный интерфейс пользователя, позволяющий изучить все средства и возможности SQL Server 2000 без написания сложных для понимания строк программного кода.

Интерфейс SQL Server 2000 во многом напоминает интерфейс стандартного Windows-приложения, включая такие элементы управления, как строки меню, пиктограммы, древовидные списки, переключатели и т.п. Благодаря этой схожести выполнять базовые задачи в SQL Server 2000 довольно быстро смогут научиться даже те пользователи, которые впервые столкнулись с этим приложением.

В этой главе приводится краткое описание компонентов SQL Server 2000, рассматриваются их основные функции и преимущества.

Следует отметить, что, хотя рассматриваемые в главе вопросы и не касаются непосредственно разработки приложения SQLSpyNet, изучение средств SQL Server 2000 является крайне важной задачей. Вы не только научитесь быстрее и эффективнее разрабатывать ориентированные на использование базы данных приложения, но и закрепите знания об SQL Server 2000.

При разработке приложения SQLSpyNet будут использованы два компонента SQL Server 2000, подробно описанные в этой главе. Эти программы раскрывают основные возможности SQL Server 2000 и обеспечивают высокую степень гибкости при разработке приложений (под гибкостью подразумевается возможность выполнить определенную задачу разными способами).

Не волнуйтесь, если при рассмотрении того или иного компонента SQL Server 2000 вы не сможете понять всю его функциональность; по мере освоения материала книги вы легко наверстаете упущенное.

Исследование объектов базы данных с помощью средства *Enterprise Manager*

Enterprise Manager является *интегрируемым приложением* консоли управления Microsoft (*Microsoft Management Console — MMC*), которая включает в себя значитель-

ную часть средств управления сетевыми серверными приложениями Windows. Вы должны были познакомиться с программой Enterprise Manager при подключении к SQL Server 2000 на этапе установки и настройки этой СУБД.

Термин *Интегрируемое приложение* (*snap-in*) — это компонент, выполняющийся внутри консоли управления Microsoft (далее MMC). Интегрируемое приложение не может выполняться само по себе; оно всегда должно находиться внутри MMC.

Существует множество различных интегрируемых приложений, предоставляющих средства управления серверными программами. Помимо SQL Server 2000, MMC поддерживает информационный сервер Internet (Internet Information Server — IIS) и сервер транзакций Microsoft (Microsoft Transaction Server — MTS), не говоря уже о том, что MMC является базой для всех остальных интегрируемых в нее приложений.

Программа SQL Server Enterprise Manager представляет собой главное управляющее приложение, предназначенное для создания и поддержки баз данных, а также для управления ими. Enterprise Manager имеет стандартный для интегрируемых приложений MMC интерфейс, что значительно облегчает пользователям, знакомым с Internet Information Server, процесс его изучения.

Предоставляемые программой Enterprise Manager средства графического интерфейса пользователя (Graphical User Interface — GUI) позволяют выполнять такие задачи, как создание резервной копии базы данных, запуск назначенного задания, создание и управление учетными записями пользователей, а также вывод различных графиков и диаграмм.

В левой панели программы SQL Server 2000 Enterprise Manager отображается древовидный список, включающий установленные на текущий момент базы данных SQL Server 2000. С помощью Enterprise Manager пользователь может не только просмотреть их названия, но и изучить содержащиеся в них объекты, включая таблицы, хранимые процедуры, пользователей и т.д.

Для того чтобы добраться до определенного объекта базы данных и получить касающуюся его детальную информацию, достаточно сделать всего несколько щелчков мышью. Таким образом, например, можно просмотреть права доступа к конкретной таблице, создать сценарий хранимой процедуры, а также удалить или переименовать представление. На рис. 2.1 изображен древовидный список объектов приложения Enterprise Manager.

Enterprise Manager позволяет быстро и легко получить информацию о процессах и объектах установленного экземпляра SQL Server 2000. Таким образом, вы можете без труда администрировать назначенные задания и выполнять другие функции по управлению базой данных, включая (но ни в коем случае не ограничиваясь) координацию безопасного доступа к серверу, импорт и экспорт информации, а также репликацию базы данных.

Основные преимущества *Enterprise Manager*

Enterprise Manager представляет собой своеобразный "центр управления" установленным экземпляром SQL Server 2000. Большинство дополнительных средств SQL Server 2000 (рассматриваемых далее в этой главе) могут быть запущены непосредственно из Enterprise Manager, включая мастер импорта и экспорта (Import and Export) и средство создания запросов (Query Analyzer).

Как упоминалось ранее, простота использования Enterprise Manager позволяет буквально "на лету" усвоить все базовые функции SQL Server 2000 и получить полный контроль над этим приложением. Enterprise Manager позволяет выполнять различные задачи администрирования, проектирования и создания базы данных.

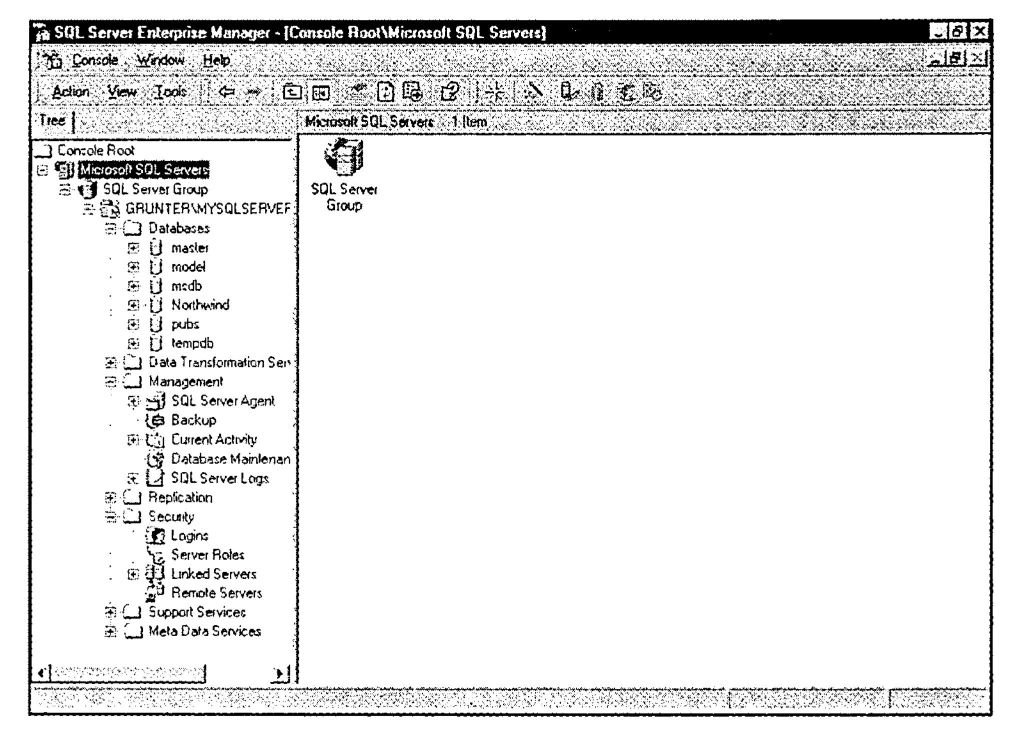


Рис. 2.1. Enterprise Manager — один из главных компонентов SQL Server 2000

В случае затруднения при поиске команды для выполнения определенного действия Enterprise Manager предлагает множество различных мастеров, призванных помочь в выполнении повседневных задач управления базой данных.

Enterprise Manager является настраиваемым приложением. Можно изменить заданное по умолчанию состояние пиктограмм (аналогично проводнику Windows), отобразить скрытые или системные объекты, а также внести коррективы в способ реагирования SQL Server 2000 на щелчок мышью (отобразить приглашение на ввод имени пользователя, запустить сервер и т.п.).

Enterprise Manager в контексте разработки приложения SQLSpyNet

Поскольку Enterprise Manager является центром управления SQL Server 2000, следует в совершенстве изучить это приложение с целью эффективного управления SQLSpyNet.

Учитывая, что Enterprise Manager позволяет выполнять как простые, так и сложные задачи администрирования, мы используем эту программу для создания базы данных приложения SQLSpyNet и построения его первых трех таблиц с применением элементов управления графического интерфейса пользователя. В главе 3, “Вербовка “виртуальных” агентов, или Создание базы данных SQLSpyNet”, рассматривается способ создания некоторых таблиц с помощью программного кода (с использованием приложения Query Analyzer).

Помимо этого, Enterprise Manager будет использоваться для определения отношений между первыми тремя созданными таблицами, что поможет убедиться в простоте создания схемы базы данных. На более поздних стадиях разработки приложения

(см. главу 11, "Администрирование разведывательной сети") будет создано несколько назначенных заданий, которые должны будут выполняться в определенное время. Выполняемые назначенными заданиями действия могут включать в себя резервное копирование базы данных, импорт и экспорт информации и даже рассылку электронных сообщений пользователям.

Как видите, значимость Enterprise Manager для разрабатываемого нами приложения весьма высока. Эта программа предоставляет не только возможность гибкого изучения средств SQL Server 2000, но и помогает расширить знания по управлению базами данных.

Выполнение запросов к базе данных с помощью средства *Query Analyzer*

Приложение Query Analyzer позволяет выполнять запросы Transact-SQL непосредственно к базе данных. С помощью этого средства можно, например, извлечь информацию из таблицы посредством оператора `SELECT`, выполнить хранимую процедуру или же создать представление (`VIEW`) через Transact-SQL. Не обращайте внимания на незнакомые названия операторов; каждый из них будет подробно рассмотрен в главах 4, "Обработка данных с помощью кода Transact-SQL", и 5, "Использование языка определения данных для просмотра и обновления информации".

Следует отметить, что этим функциональные возможности Query Analyzer далеко не исчерпываются. Результаты выполнения запроса могут быть просмотрены в форме таблицы или текста. Просмотрев результат выполнения запроса в форме таблицы, его можно сохранить в виде разделенного запятыми списка, что идеальным образом подходит для экспорта результата запроса в статистический пакет (например, Excel) с целью построения диаграммы или проведения последующего анализа.

Query Analyzer позволяет создать *план исполнения* запроса Transact-SQL. План исполнения необходим для проведения анализа запроса с целью оптимизации его наиболее ресурсоемких частей. Следует отметить, что оптимизация запроса проводится с условием сохранения всех самых важных его результатов.

Термин *План исполнения* (*execution plan*) содержит информацию о способе извлечения результатов запроса с использованием для этого таблиц и индексов. Хранящаяся в плане исполнения информация может быть представлена тремя способами: графически, в текстовом виде или в сжатом текстовом формате.

В Query Analyzer входит мастер Index Tuning Wizard, который анализирует использованные в запросе индексы. На основе проведенного анализа Index Tuning Wizard делает вывод о необходимости добавления к использованным в запросе таблицам дополнительных индексов с целью повышения производительности.

Термин *Индекс* таблицы во многом подобен предметному указателю книги. SQL Server 2000 в несколько раз быстрее находит строку таблицы с объявленным на ней индексом, нежели строку без него. По умолчанию на первичном ключе таблицы объявляется уникальный (*unique*) индекс, который однозначно определяет все его строки.

Еще одним средством, входящим в состав Query Analyzer, является синтаксический анализатор кода Transact-SQL, который проверяет синтаксис записанных выражений перед их выполнением (в этом смысле синтаксический анализатор подобен

компилятору). Использование синтаксического анализатора помогает выявить синтаксические ошибки в выражениях Transact-SQL до их выполнения.

Пожалуй, самым полезным нововведением Query Analyzer, появившимся в SQL Server 2000, является поддержка древовидного списка объектов. И хотя такой список — вещь для SQL Server не новая (он уже поддерживался в Enterprise Manager), в Query Analyzer этот элемент управления появился впервые, получив название Object Browser (Обозреватель объектов). Добавление древовидного списка объектов к интерфейсу Query Analyzer значительно упростило процесс построения запросов Transact-SQL (а значит, и жизнь разработчикам приложений).

Object Browser позволяет просматривать не только поддерживаемые конкретным сервером/экземпляром SQL Server 2000 базы данных, но и соответствующие выбранной базе данных таблицы, представления, хранимые процедуры, пользователей и т.п. На рис. 2.2 изображено окно программы Query Analyzer, в левой части которого находится древовидный список объектов Object Browser.

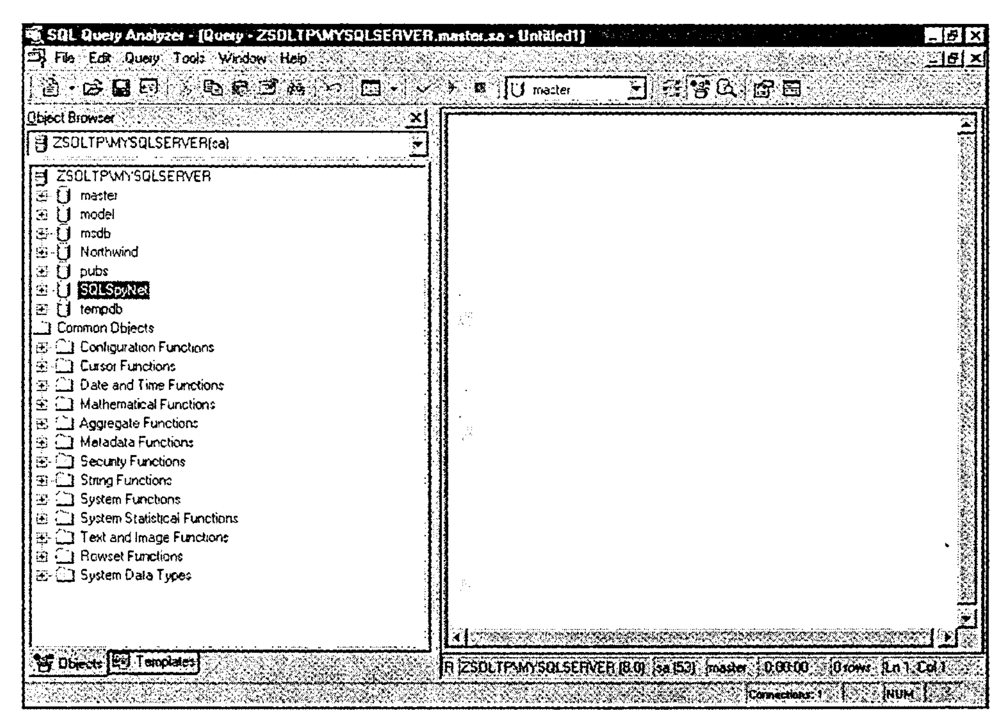


Рис. 2.2. Поддержка древовидного списка объектов Object Browser в Query Analyzer — новое свойство SQL Server 2000

Экскурс

Почему же именно Object Browser стал самым полезным нововведением программы Query Analyzer?

Для того чтобы ответить на этот вопрос, достаточно всего лишь вспомнить те действия, которые необходимо было выполнить для построения запроса Transact-SQL в предыдущих версиях *Query Analyzer*. При построении запроса приходилось постоянно переключаться в *Enterprise Manager* для того, чтобы узнать имена объектов базы данных, которые затем подставлялись в запрос. Естественно, что с появлением *Object Browser* такая необходимость исчезла раз и навсегда.

Экскурс Более того, *Object Browser* столь органично вошел в состав постоянно используемых средств *Query Analyzer*, что представить себе процесс построения запроса Transact-SQL без *Object Browser* сейчас крайне сложно

Несмотря на это, Microsoft решила пойти еще дальше и сделать процесс разработки приложения простым и увлекательным. Предложенное Microsoft новшество — поистине фантастический подарок всем разработчикам SQL Server 2000. Посудите сами: теперь, для того чтобы создать простой запрос к таблице, следует всего лишь щелкнуть на этой таблице правой клавишей и выбрать из контекстного меню одну из команд: Select (Выбрать), Insert (Вставить), Update (Обновить) или Delete (Удалить). Появится окно, содержащее сгенерированный в соответствии с выбранной командой код Transact-SQL (для запросов соответствующего типа будут также сгенерированы все типы данных). И если вам приходилось раньше писать вручную операторы SELECT для доступа к нескольким столбцам или операторы UPDATE без точного знания всех типов обновляемых полей, то вы должны по достоинству оценить это блестящее нововведение. На рис. 2.3 изображен процесс создания простого запроса к таблице.

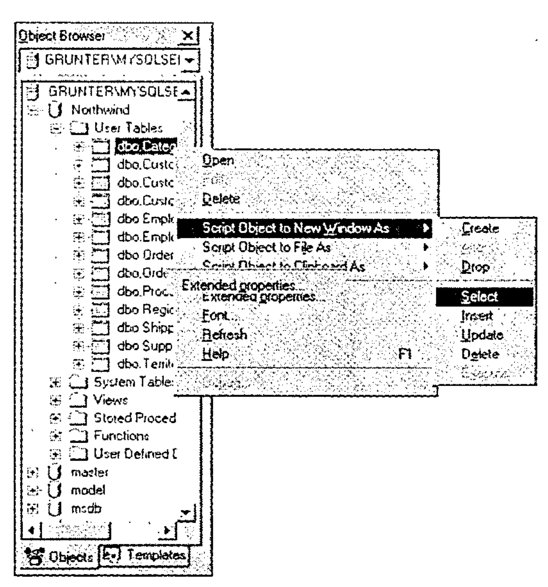


Рис. 2.3. Благодаря Microsoft процесс создания простого запроса к таблице сведен всего лишь к нескольким щелчкам мышью

Трудно поверить, но и это еще не все. Для повышения и без того достаточно "продвинутой" функциональности пользовательского интерфейса Microsoft включила в Object Browser набор самых распространенных шаблонов кода Transact-SQL.

Названия групп шаблонов говорят сами за себя: здесь находятся шаблоны создания базы данных (Create Database (Создать базу данных)), таблиц (Create Table (Создать таблицу)), хранимой процедуры (Create Stored Procedure (Создать хранимую процедуру)) и др. Для того чтобы просмотреть полный список доступных шаблонов, следует всего лишь щелкнуть на закладке Templates (Шаблоны), которая находится в нижней части окна Object Browser. Выбрав нужный шаблон, щелкните на нем дважды (или щелкните правой кнопкой мыши и выберите из контекстного меню команду Open (Открыть)), в результате чего будет создано новое окно с кодом Transact-SQL.

Все, что остается сделать, — заменить оставленные пустыми поля их реальными значениями, после чего запрос полностью готов к выполнению. На рис. 2.4 изображен пример использования шаблона Create Database.

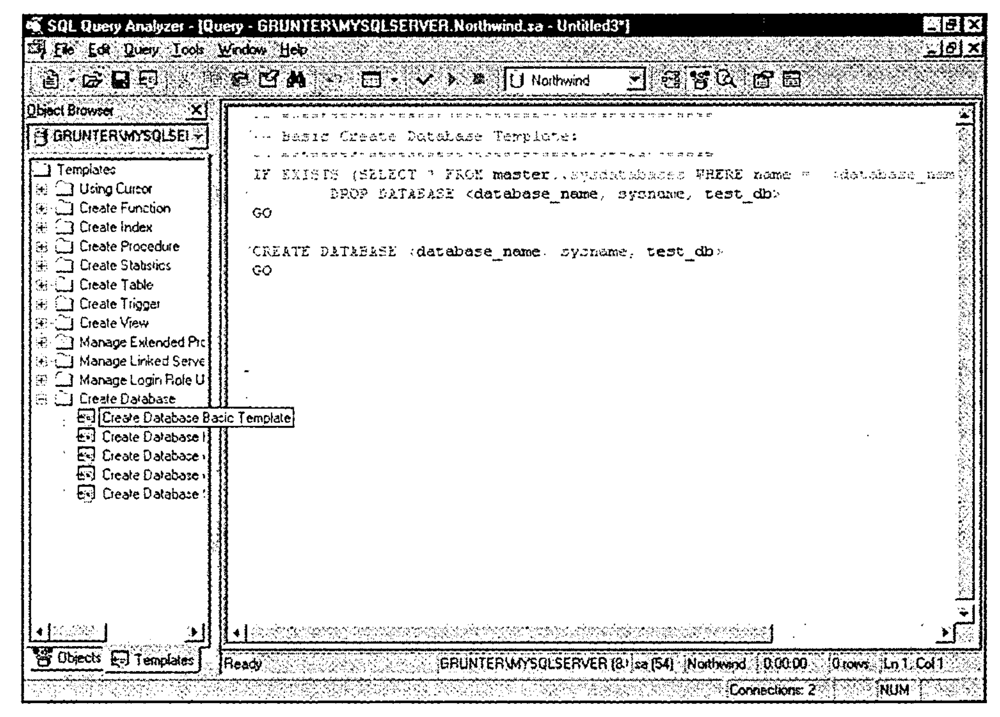


Рис. 2.4. Пример автоматического создания кода Transact-SQL в результате использования шаблона Create Database

Напоследок стоит отметить возможность редактирования стандартных шаблонов в соответствии с собственными требованиями. Вы можете либо создать абсолютно новый шаблон, который предназначается, скажем, для выполнения ежедневных задач администрирования (что характерно для большинства администраторов баз данных), либо изменить в соответствии с целями своей организации один из стандартных шаблонов, поставляемых с SQL Server 2000.

На этом список функциональных возможностей Query Analyzer не заканчивается. Приобретая все больший опыт работы, вы, несомненно, откроете для себя еще очень много возможностей этого замечательного средства SQL Server 2000, включая множество полезных и интересных функций, призванных облегчить выполнение рутинных заданий.

Действия, которые можно выполнить с помощью Query Analyzer

Query Analyzer предоставляет весьма гибкие функциональные возможности. С помощью этого средства можно выполнить практически те же самые операции, что и с помощью Enterprise Manager, с тем лишь отличием, что для выполнения операций следует использовать не графический интерфейс, а код Transact-SQL. Работая

с Query Analyzer, вы не только изучите инструментальные средства языка Transact-SQL, включая структуру и форму его запросов, но и приобретете навыки профессионального администратора базы данных.

Query Analyzer поддерживает возможность компиляции запроса Transact-SQL перед его выполнением. Это свойство является одним из наиболее полезных, так как позволяет убедиться в синтаксической правильности кода перед его передачей на обработку механизму выполнения запросов базы данных. Более того, Query Analyzer поддерживает цветовую разметку программного кода Transact-SQL для более наглядного представления ключевых слов и специальных функций SQL Server 2000. Цветовая гамма полностью настраиваемая, что делает возможным создание собственных комбинаций цветов.

Query Analyzer позволяет анализировать процесс выполнения запроса посредством плана исполнения. План исполнения необходим для оценки эффективности использования индексов и других доступных ресурсов базы данных.

И наконец, одно из последних, но не менее важных преимуществ использования Query Analyzer — независимость от графического интерфейса пользователя, что делает возможным выполнение всех задач по управлению базой данных даже в случае недоступности графического интерфейса (что может случиться, например, из-за некорректной регистрации какой-нибудь библиотеки `.dll`).

Использование *Query Analyzer* в контексте разработки приложения SQLSpyNet

Средство построения запросов Query Analyzer весьма активно используется на протяжении всего процесса разработки приложения SQLSpyNet. И хотя, как упоминалось ранее, для создания базы данных и трех ее основных таблиц будет применяться Enterprise Manager, создание всех оставшихся таблиц, равно как и проверка корректности кода Transact-SQL, будет целиком возложено "на плечи" Query Analyzer.

Query Analyzer используется для изучения операторов Transact-SQL (см. главы 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", и 4, "Обработка данных с помощью кода Transact-SQL"), а также для построения абсолютного большинства применяемых в приложении хранимых процедур.

Как отмечалось выше, Query Analyzer — это весьма гибкое средство, входящее в поставку SQL Server 2000. Интегрируя в себе некоторые другие компоненты, Query Analyzer превращается в многофункциональное и эффективное средство разработки базы данных и управления ею.

"Слежение" за выполняемыми базой данных действиями с помощью программы *SQL Profiler*

Входящая в состав SQL Server 2000 программа SQL Profiler предназначена для выявления слабых мест в текущей конфигурации базы данных. SQL Profiler позволяет создать *файл трассировки* для операторов Transact-SQL, которые передаются на выполнение механизму выполнения запросов SQL Server 2000.

Термин В *файле трассировки* фиксируются все выполняемые на сервере базы данных действия, для того чтобы их можно было впоследствии просмотреть или выполнить заново. Как правило, файл трассировки сохраняется для последующего рассмотрения, однако иногда возникает необходимость срочно воссоздать на его основе последовательность произошедших в системе событий.

Для того чтобы запустить программу SQL Profiler, выполните одно из следующих действий:

- в Enterprise Manager выполните команду Tools⇨SQL Profiler (Сервис⇨SQL Profiler);
- выберите команду Start⇨Programs⇨Microsoft SQL Server⇨Profiler (Пуск⇨Программы⇨Microsoft SQL Server⇨Profiler) или любую другую команду, обозначающую группу SQL Server 2000.

Одним из наиболее распространенных способов использования программы SQL Profiler является просмотр запросов Transact-SQL, отправленных на выполнение базе данных клиентским приложением. В соответствующем файле трассировки будут зафиксированы такие действия, как вызов хранимых процедур, передача параметров, создание представлений, а также вход пользователей в систему базы данных с помощью клиентского приложения. На рис. 2.5 показано окно программы SQL Profiler.

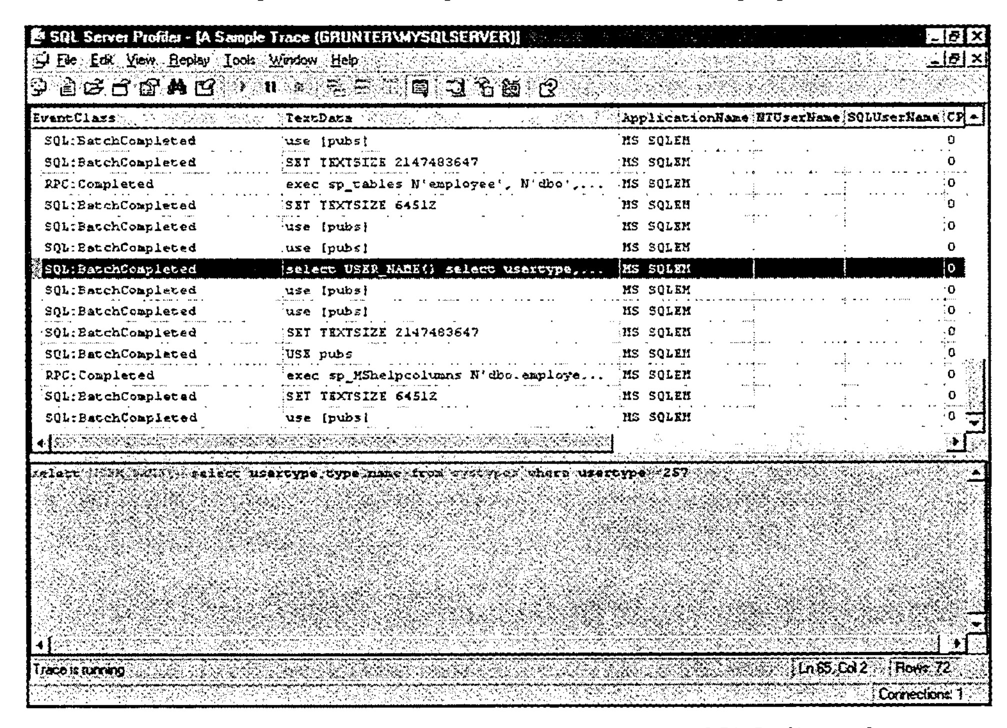


Рис. 2.5. Просмотр файла трассировки с помощью программы SQL Profiler — одного из основных компонентов SQL Server 2000

Одна из наиболее сильных сторон программы SQL Profiler — возможность отслеживать выполнение операторов Transact-SQL, что позволяет выявлять сложные или медленно выполняющиеся запросы и оптимизировать их с целью повышения произ-

водительности (уменьшения времени выполнения). Помимо этого, SQL Profiler предназначена для текущего контроля за выполнением приложения, в частности за использованием доступных ресурсов.

Следует отметить, что простота применения графического интерфейса пользователя позволяет даже новичкам без труда создавать собственные файлы трассировки и на их основе выполнять диагностику различных проблем.

Ситуации, в которых следует использовать программу *SQL Profiler*

Программа SQL Profiler предназначена для выяснения причины сбоев в работе взаимодействующих с базой данных приложений.

Экскурс

***Обновление SQL Server и программа* SQL Profiler**

Программа *SQL Profiler* может оказаться очень полезной при переходе со старой версии SQL Server на новую (например, с версии 6.5 на версию 7.0). Представим себе, что один из клиентов все еще использует довольно старое приложение, написанное на Visual Basic сторонним разработчиком; при этом возможность просмотреть исходный код данного приложения отсутствует. Единственным способом, с помощью которого в подобной ситуации можно убедиться в успешности перехода на более новую версию SQL Server, является анализ операторов Transact-SQL, посылаемых этим приложением на выполнение базе данных, т.е. как раз то, с чем легко справляется программа *SQL Profiler.*

Программа SQL Profiler выгодно отличается гибкостью своих функциональных возможностей, позволяя одинаково легко получать обзор ситуации всех компонентов сервера. Созданный файл трассировки может быть сохранен на диске для последующего использования. При необходимости область трассировки может быть сужена до масштабов отдельного пользователя или компьютера и, напротив, расширена до масштабов целого сервера.



При сохранении и воспроизведении файлов трассировки следует проявлять осторожность, что особенно касается "реальных" баз данных. Причина очень проста: поскольку все операторы Transact-SQL будут выполняться второй раз, это может привести к повторному внесению одних и тех же изменений в базу данных. Подобная практика может стать причиной неожиданных сбоев в работе большинства баз данных, функционирующих в режиме реального времени.

Использование программы *SQL Profiler* в контексте разработки приложения SQLSpyNet

Так в какой же ситуации придется воспользоваться программой SQL Profiler для отслеживания деятельности тайных агентов и анализа посылаемых ими шифрограмм? Будем надеяться, что такая ситуация не наступит никогда. Мы собираемся разработать столь эффективный и надежный код, выполнение которого ни в коем случае не должно привести к появлению ошибок. (На самом деле мы все-таки обратимся к программе SQL Profiler в главе 13. "Сбор разрозненных данных в один источник").

И все же следует отметить, что даже при самом что ни на есть тщательном подходе к разработке приложения гарантировать полное отсутствие в нем ошибок было бы слишком опрометчиво. Вряд ли существует хотя бы одна программа, на 100% свободная от "жучков". И если кто-то утверждает, что он видел такую программу, то, держу пари, это была либо программа, состоящая из одной строки кода, либо этот человек просто не успел заметить ошибки в ее работе. SQL Profiler позволяет выявить самые незначительные сбои в коде программы, а также обеспечивает отслеживание всех выполняемых действий на уровне приложения, т.е. можно оценить производительность SQL Server 2000.

Чтобы получить максимум преимуществ, прежде всего следует научиться эффективно использовать программу SQL Profiler. Как правило, она необходима лишь в тех ситуациях, когда в приложении появляется трудно диагностируемая ошибка или возникает потребность в отслеживании всех выполняемых базой данных действий. В подобных случаях всегда приятно сознавать, что под рукой есть эффективное и надежное средство, которое поможет справиться с любого рода неприятностями.

Импорт и экспорт данных с помощью средства *Data Transformation Services (DTS)*

Щелчок на пиктограмме Import and Export Data (Импорт и экспорт данных), расположенной в меню Start⇨Programs (Пуск⇨Программы) в группе Microsoft SQL Server, приводит к запуску мастера преобразования данных — Data Transformation Services (DTS) Wizard.

Подобно программе SQL Profiler, для запуска мастера DTS Import/Export Wizard выполните одно из следующих действий:

- в Enterprise Manager выберите команду Tools⇨Data Transformation Services (Сервис⇨Службы преобразования данных), а затем команду Import (Импорт) или Export (Экспорт);
- выберите команду Start⇨Programs⇨Microsoft SQL Server⇨Import and Export Data (Пуск⇨Программы⇨Microsoft SQL Server⇨Импорт и экспорт данных) или любой другой пункт, обозначающий группу пиктограмм SQL Server 2000.

Мастер DTS Import/Export Wizard позволяет выполнять импорт и экспорт информации из базы данных SQL Server 2000, используя для этого такие источники, как ODBC и OLE DB (рис. 2.6).

Мастер DTS Import/Export Wizard обеспечивает обмен информацией со многими разнородными источниками данных, в том числе:

- базами данных (Oracle, Microsoft Access, IBM DB2, SQL Server и др.);
- электронными таблицами;
- текстовыми файлами.

Следует отметить, что выполнение операции обмена данными не составляет большого труда, поскольку DTS Import/Export Wizard включает в себя достаточное количество шагов, предназначенных для "устранения" вмешательства пользователя в процессе обмена информацией.

Функциональность DTS Import/Export Wizard не ограничивается только обменом информацией; так, например, в процессе извлечения информации из другой базы данных можно импортировать *целиком* всю ее таблицу. Таким образом, мастер преобразования данных позволяет извлекать из внешней базы данных не только хранящуюся в ней информацию, но и элементы структуры составляющих ее таблиц (имена столбцов, типы данных и т.д.).

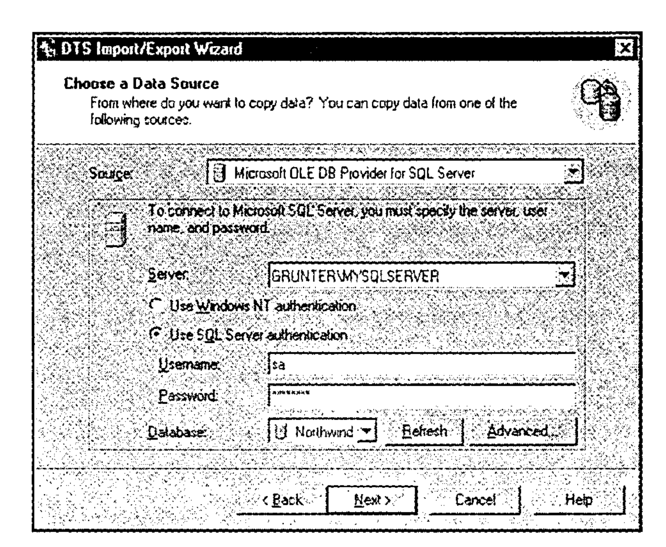


Рис. 2.6. Диалоговое окно мастера DTS Import/Export Wizard, входящего в поставку SQL Server 2000

Завершив ввод всей необходимой информации, следует указать мастеру DTS Import/Export Wizard точное время проведения операции по обмену данными (она может быть проведена немедленно или отложена до наступления заданного времени).

Назначение операции по обмену данными на более позднее время как правило бывает связано либо со слишком большой загрузкой сервера в текущий момент, либо с необходимостью дождаться получения информации от источника (время передачи которой обычно оговаривается заранее). Обмен информацией в таком случае лучше всего начинать после прибытия свежих данных, так как это поможет уменьшить объем действий, выполняемых вручную.

Основные преимущества *DTS*

DTS предоставляет наглядный и интуитивно понятный интерфейс для выполнения операций импорта и экспорта информации из базы данных SQL Server 2000.

Как упоминалось ранее, DTS поддерживает возможность импорта и экспорта информации из множества различных источников данных. Благодаря особой организации (напомним, что DTS представляет собой мастер) DTS позволяет легко определить начальный и конечный источник передачи информации.

DTS обладает определенной гибкостью, состоящей в том, что операция по передаче информации может быть проведена как незамедлительно, так и по прошествии заданного промежутка времени. Более того, сведения о выполняемой операции передачи данных могут быть сохранены на диске, что позволяет впоследствии изменять их в соответствии с текущими задачами.

Вооружившись богатым набором функциональных возможностей, предлагаемых DTS, пользователь может создавать собственные сценарии, отправлять сообщения электронной почты, отменять только что внесенные в базу данных изменения, планировать назначенные задания, а также выполнять еще множество полезных и нужных задач. Благодаря тому огромному вкладу, который привносит DTS в процесс импорта и экспорта информации, можно не только управлять SQL Server 2000 с помощью множества различных средств, но и контролировать входящие и исходящие потоки информации.

Использование *DTS* для импорта информации в SQLSpyNet

Для того чтобы наглядно продемонстрировать способ работы мастера DTS Import/Export Wizard, создадим несколько электронных таблиц Excel, содержащих информацию об именах тайных агентов, данные о злоумышленниках, а также другие сведения, после чего импортируем эти данные в разрабатываемое приложение. Сохранив соответствующий сценарий DTS на диске, просмотрим полученный пакет с помощью Enterprise Manager. (И все же стоит отметить, что наиболее "серьезные" функции мастера DTS Import/Export Wizard рассматриваются в главе 14, "Отладка и устранение ошибок в SQL Server 2000".)

Расширив масштаб нашей деятельности, воспользуемся DTS для импорта других подобных данных. Держу пари, что перевод информации, хранящейся на старых добрых мейнфреймах, в суперсовременный формат базы данных SQL Server 2000 позволит проводить наши тайные операции на невиданном доселе уровне!

Обзор остальных компонентов SQL Server 2000

В предыдущих разделах рассмотрены лишь некоторые компоненты SQL Server 2000. Поскольку SQL Server 2000 является полнофункциональной СУБД, в ее составе имеется еще достаточно средств, предназначенных для настройки и управления. Ничуть не умаляя значимость этих средств для SQL Server 2000 и особенно для администраторов баз данных, мы вынуждены оставить их за пределами нашего обсуждения. В конце концов, цель данной книги заключается в разработке реального приложения, а не в скрупулезном изучении отдельных компонентов SQL Server 2000!

Несмотря на это, остаток данного раздела посвящен краткому обзору средств SQL Server 2000 оказавшихся за пределами детального рассмотрения. Впоследствии вам наверняка захочется познакомиться с ними поближе для того, чтобы повысить навыки администратора баз данных. Некоторые из этих средств также описываются в приложении В, "Ну и что дальше?".

- Сервисная программа Server Network Utility позволяет выполнять настройку и вносить изменения в состав сетевых библиотек SQL Server 2000, установленных на данном компьютере.
- Сервисная программа Client Network Utility позволяет выполнять настройку сетевых библиотек, которые используются клиентом для подключения к SQL Server 2000.
- Service Manager — это небольшое сервисное приложение, которое разрешает запуск, приостановку и остановку процесса выполнения отдельных служб SQL Server 2000. Данная программа позволяет работать с любым экземпляром SQL Server 2000, установленным на сервере, а также с любыми сетевыми экземплярами SQL Server 2000. Service Manager предоставляет возможность управления несколькими службами.
    - Служба *SQL Server*. Запускает установленный на компьютере сервер базы данных. Если служба SQL Server не запущена, ни один клиент не сможет подключиться к экземпляру SQL Server 2000.

- Служба *SQL Server Agent.* Запускает все административные задачи, которые выполняются в пределах данного экземпляра SQL Server 2000 (например, назначенные задания).
- Служба *Microsoft Search.* Запускается только на компьютерах под управлением операционных систем Windows NT и Windows 2000 и предоставляет средства полного текстового поиска.
- Служба *MSDTC.* Представляет собой координатор распределенных транзакций.

- Программа Analysis Services позволяет создавать хранилища данных, которые впоследствии будут использованы для анализа. Благодаря Analysis Services удается снизить нагрузку на базу данных со стороны клиентских запросов.
- English Query представляет собой еще одно дополнительное средство SQL Server 2000, позволяющее пользователям обращаться к базе данных с вопросами, напоминающими стандартные фразы английского языка. Таким образом, English Query дает возможность извлекать информацию из базы данных даже тем пользователям, кто не имеет ни малейшего представления о языке SQL.

Резюме

В данной главе описаны основные компоненты SQL Server 2000. Следует отметить, что спектр различных средств SQL Server 2000 весьма обширен и предоставляет множество функциональных возможностей. С помощью этих средств можно получить полный контроль над базой данных, хранящейся в ней информацией и сервером без написания длинных и сложных операторов программного кода.

С помощью предоставляемых компонентами SQL Server 2000 интерфейсов может быть выполнено большинство задач по управлению базой данных.

Несмотря на то что в этой главе представлено лишь краткое описание компонентов SQL Server 2000, в ходе изучения материала следующих глав вы сможете познакомиться с ними поближе.

Следующие шаги

В процессе изучения материала следующей главы мы наконец-то создадим базу данных SQLSpyNet, причем с использованием не только Enterprise Manager, но и Query Analyzer. Таким образом, у вас появится прекрасный шанс познакомиться со всеми "черновыми" аспектами разработки базы данных в SQL Server 2000!

Глава 3

Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet

В этой главе...

Дамы и господа! Вот наконец-то и наступил тот долгожданный момент, о котором вскользь упоминалось в двух предыдущих главах. Без сомнения, для нас с вами (ну, по крайней мере, для меня) это очень важный момент, так как именно с него начинается многообещающее "путешествие" в мир SQL Server 2000. В третьей главе описывается процесс создания базы данных приложения SQLSpyNet и проектирование ее основных таблиц. Будут рассмотрены такие важные концепции теории баз данных, как нормализация и целостность. Помимо этого, будет завершена разработка диаграммы отношений между объектами (ERD) и подведены итоги анализа приложения.

Однако, прежде чем приступить к выполнению перечисленных задач, следует провести несколько процедур по администрированию базы данных, вызванных необходимостью убедиться в соответствии настроек SQL Server 2000 предъявляемым к приложению требованиям.

По сути, глава состоит из двух частей, что несколько увеличивает ее объем, однако в процессе перенесения объектов диаграммы отношений между объектами в базу данных вы заметите, что изложенный в этих частях материал тесно взаимосвязан. Наряду с основной темой, в данной главе рассматриваются некоторые интересные подробности, касающиеся реляционной теории, а также обсуждается назначение параметров настройки, использование которых считается "хорошим тоном" при проектировании базы данных. Материал этой главы может послужить наглядным пособием при создании вашего следующего проекта.

Администрирование *SQLSpyNet*

Прежде всего следует убедиться в том, что наше приложение (а равно и SQL Server 2000) защищено от несанкционированного доступа. Кроме того, не мешает получить некоторое представление о принципах управления учетными записями пользователей.

Не обладая способностью к телепатии, я не могу знать пароль, назначенный вами при установке SQL Server 2000 учетной записи sa, однако могу предположить, что это либо пустая строка, либо что-то наподобие password.

Термин Пользователь с учетной записью sa (System Administrator — системный администратор) имеет неограниченные полномочия при работе с SQL Server 2000. Зарегистрировавшись под именем sa, пользователь получает полный контроль над экземпляром SQL Server 2000 и всеми установленными базами данных. Будьте осторожны — обладание слишком большими возможностями может сыграть с вами злую шутку!

Вне зависимости от назначенного учетной записи sa пароля, практика его изменения поможет лучше познакомиться с системой безопасности SQL Server 2000 и приобрести весьма ценный навык, который достаточно часто приходится использовать администраторам базы данных в том случае, если какой-нибудь пользователь забыл свой пароль (а это, к сожалению, практически неизбежно).



Говоря об установленном экземпляре SQL Server 2000, я предполагаю, что эта установка была выполнена либо в соответствии с указаниями, приведенными в приложении Б, "Установка и настройка SQL Server 2000", либо каким-то другим подобным образом. Одним из наиболее важных моментов при этом является способ подключения к серверу SQL Server 2000, а потому для большей уверенности советую вам еще раз просмотреть указания по установке SQL Server 2000, приведенные в приложении Б.

Если SQL Server 2000 установлен на компьютер под управлением операционной системы Windows NT или Windows 2000, следует убедиться в том, что вы используете учетную запись с именем sa, а не administrator. В том случае, если администратор сети не разрешит вам экспериментировать с SQL Server 2000, установите экземпляр SQL Server 2000 Personal Edition на отдельный компьютер.

Установка пароля для учетной записи sa

Чтобы установить пароль для учетной записи sa, запустите приложение Enterprise Manager и выполните ряд действий.

1. Воспользовавшись деревом объектов, расположенным в левой части Enterprise Manager, отыщите папку Security (Безопасность). В данной папке содержится информация, касающаяся системы безопасности SQL Server 2000, включая учетные записи пользователей базы данных (такие, как sa), серверные роли (например, *System Administrators*), а также связанные и удаленные серверы (позволяющие подключаться к другим серверам).

   Папка Security содержит пиктограмму Logins (Учетные записи).

2. Щелкните на пиктограмме Logins для того, чтобы просмотреть учетные записи, определенные для данного экземпляра SQL Server 2000 (а также для всех существующих баз данных). Если параметры настройки вашего экземпляра SQL Server 2000 совпадают с параметрами настройки экземпляра, использовавшегося при написании этой книги, то вы увидите всего лишь одну учетную запись sa (рис. 3.1).

   Щелкните дважды на пиктограмме, соответствующей учетной записи sa. На экране появится диалоговое окно свойств учетной записи, изображенное на рис. 3.2.

Рис. 3.1. Учетные записи пользователей, определенные для данного экземпляра SQL Server 2000

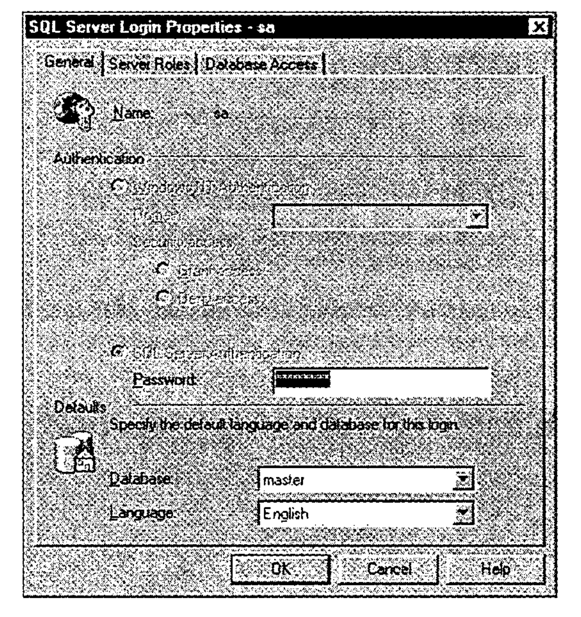


Рис. 3.2. Свойства учетной записи sa

3. В соответствующем поле ввода диалогового окна свойств учетной записи sa введите новый пароль (при выборе пароля лучше всего пользоваться правилом, в соответствии с которым пароль должен быть достаточно простым для запоминания и достаточно сложным для того, чтобы его могли случайно угадать другие), после чего щелкните на кнопке Apply (Применить) или на кнопке OK. После ввода нового пароля система автоматически попросит вас подтвердить внесенные изменения.



Единственным недостатком при изменении пароля учетной записи sa является необходимость редактирования используемых свойств регистрации сервера.

4. Выделите объект, соответствующий копии SQL Server 2000 (он расположен под серверной группой, созданной при установке SQL Server 2000 в соответствии с указаниями, приведенными в приложении Б, "Установка и настройка SQL Server 2000"), щелкните на нем правой кнопкой мыши и выберите из появившегося контекстного меню команду Edit SQL Server Registration Properties (Редактировать свойства регистрации SQL Server). На экране появится диалоговое окно, напоминающее окно первичной регистрации (рис. 3.3).

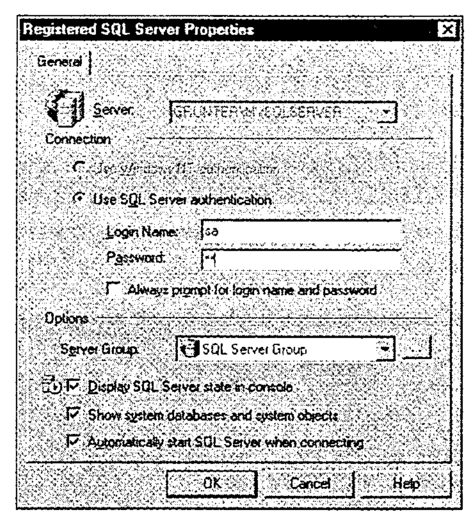


Рис. 3.3. Диалоговое окно свойств регистрации SQL Server 2000

5. Введите новый пароль и щелкните на кнопке OK. SQL Server 2000 спросит о необходимости изменения информации о текущем подключении к серверу, как показано на рис. 3.4.

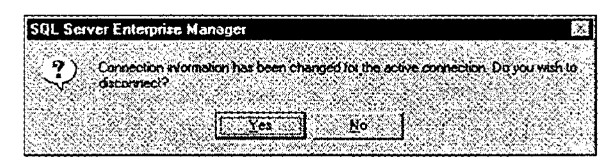


Рис. 3.4. Диалоговое окно, предлагающее подтвердить необходимость изменения информации о текущем подключении к серверу

6. Получив подтверждение, SQL Server 2000 обновит информацию, которая будет использоваться при подключении к SQL Server 2000 с помощью какого-либо клиентского средства.

На заметку Обновляя информацию о соединении, SQL Server 2000 сначала разрывает текущее соединение с сервером, а затем восстанавливает его.

При наличии какой-либо зависящей от текущего сеанса информации, например о способе обработки значений `NULL`, существует вероятность ее потери. Поэтому при восстановлении соединения с сервером следует сразу же проверить все параметры настройки.

Подводя итог, можно отметить, что изменение пароля — довольно простая задача. К тому же установка нового пароля для учетной записи обычного пользователя не требует изменения информации о текущем соединении.

Вторая задача по администрированию, которую необходимо выполнить, — проверка соответствия параметров настройки базы данных `model` предъявляемым к разрабатываемому приложению требованиям.

Термин База данных `model` — это системная база данных, используемая SQL Server 2000 в качестве шаблона при создании новой базы данных. Помимо `model`, SQL Server 2000 имеет еще несколько системных баз данных, например `master` и `tempdb`.

Настройка параметров базы данных `model` в соответствии с предъявляемыми к приложению *SQLSpyNet* требованиями

База данных `model` имеет несколько основных параметров настройки, копируемых при создании новой базы данных, которое осуществляется с помощью либо средства Enterprise Manager (используя графический интерфейс пользователя), либо Query Analyzer (используя язык запросов Transact-SQL). Для того чтобы получить доступ к параметрам настройки базы данных `model`, выполните описанные ниже действия.

1. После установки подключения с SQL Server 2000 отыщите базу данных `model` с помощью дерева объектов Enterprise Manager. Наша задача — настроить параметры базы данных `model` таким образом, чтобы создаваемые на ее основе новые базы данных наиболее оптимально соответствовали предъявляемым к приложению SQLSpyNet требованиям.
2. Выделив базу данных `model`, щелкните на ней правой кнопкой мыши и выберите из контекстного меню команду Properties (Свойства). На экране появится диалоговое окно, подобное показанному на рис. 3.5.

   Во вкладке General (Общие) находятся основные сведения о настройках параметров базы данных `model`.

На заметку В большинстве случаев единственным параметром базы данных `model`, который необходимо изменить, является параметр, определяющий начальный размер выделяемого для новой базы данных дискового пространства. В конце концов, предлагаемые SQL Server 2000 настройки по умолчанию не настолько плохи, чтобы от них полностью отказываться.

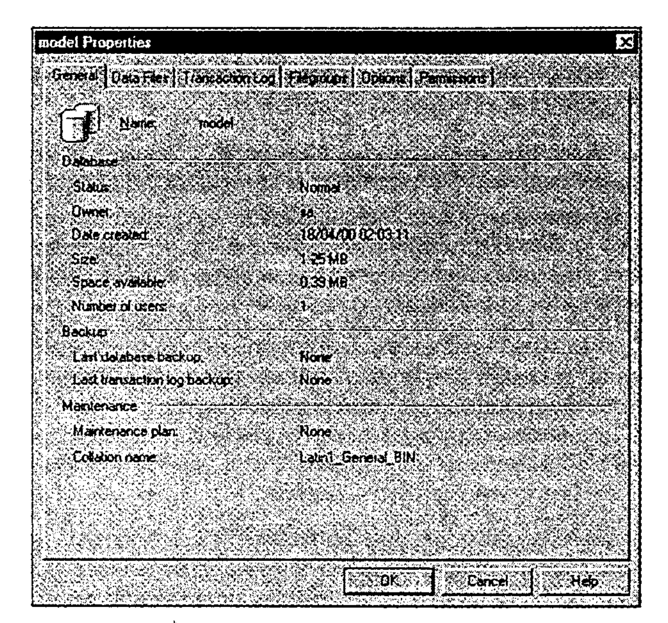


Рис. 3.5. Диалоговое окно, содержащее свойства базы данных `model`

В процессе изменения настроек параметров базы данных `model` необходимо тщательно проанализировать содержимое всех вкладок диалогового окна свойств. Это будет сделано в три этапа:

- проверка выделяемого для базы данных `model` дискового пространства;
- настройка основных параметров базы данных `model`;
- проверка прав доступа к базе данных `model`.

Чтобы сделать выполняемые действия более понятными, ниже приводится детальное описание всех вкладок диалогового окна свойств базы данных `model`.

Файловое пространство

Прежде всего следует проверить настройки, касающиеся выделяемого под `model` (а следовательно, и под любую новую базу данных, создаваемую на основе шаблона базы данных `model`) дискового пространства. Активизируйте вкладку Data Files (Файлы данных), показанную на рис. 3.6.

Вкладка Data Files содержит параметры настройки, касающиеся выделяемого под базу данных дискового пространства. Как видно из рис. 3.6, в конфигурации, которая использовалась при написании этой книги, все файлы базы данных хранятся на диске D. Основной причиной такого решения послужил факт, что операции ввода-вывода являются наиболее узким местом при разработке практически любых приложений.

Экскурс

Некоторые "узкие места", встречающиеся при разработке баз данных

Несмотря на то что объем жестких дисков постоянно растет, а скорость считывания хранящейся на них информации постепенно увеличивается, сам метод доступа к этой информации остается практически неизменным на протяжении вот уже более 20 лет.

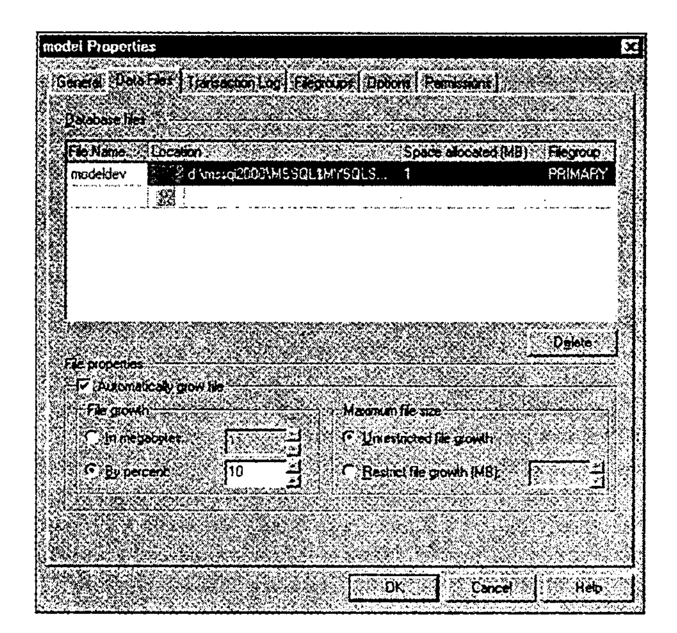


Рис. 3.6. Параметры настройки дискового пространства

Экскурс

Когда СУБД запрашивает очередную порцию информации, находящаяся внутри жесткого диска головка (которую, по сути, можно сравнить с головкой граммофона) сканирует его в поиске нужных данных. Теперь представьте себе ситуацию, в которой к базе данных подключено 200 пользователей, одновременно требующих возврата нужной им информации. Стоит ли говорить, что "чувствует" при этом головка жесткого диска! Ведь ей требуется извлекать, обновлять и поддерживать записываемые на жесткий диск данные со скоростью, достаточной для обеспечения комфортной работы всех 200 (а вообще говоря, любого количества) пользователей. Наиболее разумно в сложившейся ситуации разделить нагрузку между несколькими жесткими дисками.

Один из наиболее эффективных методов для достижения этой цели — использование *RAID-массива* уровня 0, который также носит название *Data Striping* (распределение данных).

Термин

RAID-массив (Redundant Array of Independent Disks — массив независимых дисковых накопителей с избыточностью) представляет собой систему, состоящую из нескольких дисковых накопителей, которая позволяет добиться более высокой производительности, надежности и в то же время увеличить объем дискового пространства при минимальных затратах. Более того, RAID-массивы обладают встроенной возможностью сохранять работоспособность при отказе некоторых дисков, что, несомненно, делает их выбор весьма привлекательным. Всего существует шесть уровней RAID-массивов (с 0 по 5).

Экскурс

Несмотря на то что RAID-массив не является частью SQL Server 2000 (это не более чем просто достаточно интересная аппаратная реализация носителя данных), RAID-массивы уровней 0, 1 и 5 очень часто используются вместе с SQL Server 2000 для повышения производительности этой СУБД.

К сожалению, RAID-массивы не поддерживаются операционными системами Windows 95 и Windows 98.

Экскурс Несколько жестких дисков, объединенных в RAID-массив, представляют собой одно целое, выполняя при этом одновременно множество различных операций.

Но что же делать пользователям, не имеющим возможности заполучить в свое распоряжение RAID-массив? При наличии нескольких жестких дисков можно разнести критически важную информацию, например файлы данных и журналы транзакций, по различным физическим накопителям. Ну а если и этот вариант не осуществим, тогда остается разве что затянуть потуже ремень и вспомнить, что народная мудрость учит "довольствоваться малым".

Для того чтобы обезопасить себя от нежелательной ситуации, связанной с нехваткой дискового пространства, следует изменить несколько параметров, расположенных во вкладке Data Files. Убедитесь в том, что флажок Automatically Grow File (Автоматическое приращение файла данных) установлен, а значение параметра File Growth By percent (Приращение файла в процентах) равно 10.

Что же означают упомянутые выше опции? Параметр Automatically Grow File указывает SQL Server 2000 на необходимость увеличения размера файла данных на 10% всякий раз, когда его размер будет приближаться к критическому значению (например, если размер файла приблизится к критической отметке в 5 Мбайт, он будет автоматически увеличен на 500 Кбайт, т.е. на 10%).

В качестве альтернативы можно воспользоваться параметром File Growth In megabytes (Приращение файла в мегабайтах), однако с этим выбором связаны такие нежелательные проблемы, как слишком быстрый выход за пределы дискового пространства и недостаточный прирост файла данных. Например, при достижении критической отметки в 5 Мбайт размер файла будет автоматически увеличен до 6 Мбайт, что может привести к возникновению ошибки в том случае, если на жестком диске остались свободными ровно 6 Мбайт дискового пространства.

В другом случае, если текущий размер файла данных составляет 300 Мбайт, то его увеличение на 1 Мбайт вряд ли даст желаемый результат. А вот если бы приращение такого файла составило 10% (30 Мбайт), это вполне позволило бы базе данных "продержаться" еще некоторое время без увеличения размера файла данных.

С помощью параметров настройки дискового пространства можно задать максимальный размер файла данных, ограничив таким образом предел его прироста. Если размер файла данных не будет ничем ограничен, то это в конце концов приведет к нехватке свободного места на диске. В том случае, если вы уверены, что размер базы данных никогда не превысит определенного значения, или же вам просто необходимо ограничить ее размер, следует явно указать количество мегабайтов, определяющее верхний предел для размера файла данных.

Ниже приведено краткое описание оставшихся параметров вкладки Data Files.

- В столбце File Name (Имя файла) содержится имя, используемое SQL Server 2000 для внутреннего обращения к области хранения данных. Начальный размер дискового пространства, выделяемого под файл данных, указывается в столбце Space Allocated (MB) (Выделенное дисковое пространство, Мбайт). Как показано на рис. 3.6, значение по умолчанию для начального размера равно 1 Мбайт, что, мягко говоря, не очень-то много. Увеличьте начальный размер дискового пространства, выделяемого под файл данных, хотя бы до 5 Мбайт. Для того чтобы это сделать, щелкните на соответствующем поле и введите требуемый размер в мегабайтах.
- В столбце Filegroup (Группа файлов) содержится имя группы, к которой принадлежит файл данных. При создании базы данных неявно создается группа файлов с именем `PRIMARY`. В нее сразу же попадает первичный файл данных, а также все остальные файлы, не отнесенные больше ни к какой другой группе.

Группа файлов PRIMARY содержит также все системные таблицы. Одной из причин увеличения начального дискового пространства, выделяемого под файл данных, была необходимость гарантии того, что группа файлов PRIMARY не выйдет за пределы выделенного ей места на диске. Если бы это случилось, в системные таблицы нельзя было бы добавить новые записи, что привело бы к возникновению ошибки. Подобная ситуация означала бы фактически крах любого приложения, так как было бы невозможно создать ни одного нового пользователя, ни одной таблицы, хранимой процедуры и т.п.

Журнал транзакций

Для того чтобы перейти к настройкам параметров журнала транзакций, следует щелкнуть на вкладке Transaction Log (Журнал транзакций), которая показана на рис. 3.7. Внешне она напоминает вкладку Data Files.

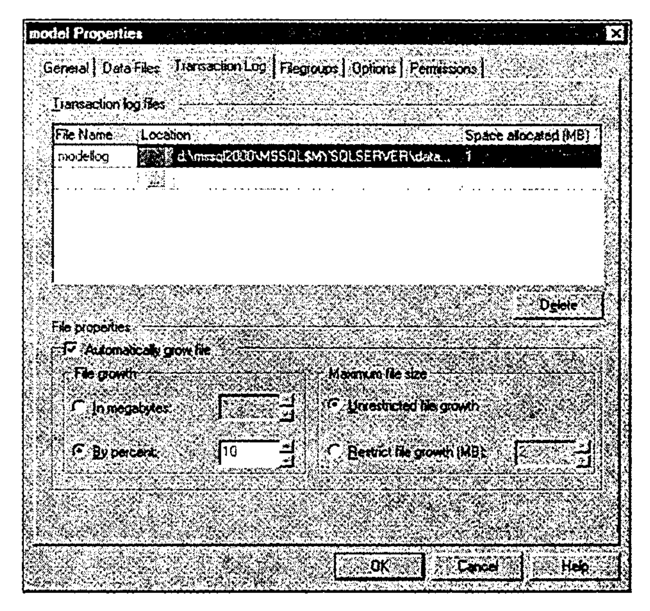
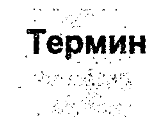


Рис. 3.7. Параметры журнала транзакций

Фактически единственным отличием настроек журнала транзакций от настроек файлового пространства является отсутствие параметра, определяющего группу файлов, к которой принадлежит журнал транзакций. Аналогично настройкам файла данных, следует установить размер дискового пространства, выделяемого под журнал транзакций, равным 5 Мбайт.

Давайте ненадолго прервемся и рассмотрим подробнее, что же представляют собой файл данных и журнал транзакций. Когда в базу данных вносится новая информация, она сохраняется в файле данных, физически расположенном на некотором дисковом носителе. Именно этот факт принципиальным образом отличает базу данных от приложения, написанного на C++ или на каком-либо другом языке программирования, поскольку база данных позволяет незамедлительно перевести внесенную в нее информацию в *перманентное* состояние.



Приложения, написанные на языке C++ или Visual Basic, хранят данные в оперативной памяти, что делает их доступными только во время выполнения программы. При использовании базы данных внесенная в нее информация незамедлительно записывается на дисковый носитель, позволяя осуществлять доступ к ней практически в любое время. Состояние, когда данные не зависят от выполняющейся программы, называется *перманентным состоянием данных* (*persisting data*).

В журнале транзакций фиксируются все изменения, внесенные в хранящуюся в базе данных информацию. Подобный "список событий" позволяет при необходимости отменить внесенное изменение (на языке баз данных — совершить *откат*). Более подробная информация об откате транзакции приводится в главе 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок". Как правило, размер журнала транзакций не бывает слишком большим, однако это зависит от конкретной конфигурации системы.

Группы файлов

Рассмотрим параметры, расположенные во вкладке Filegroups (Группы файлов), как показано на рис. 3.8.

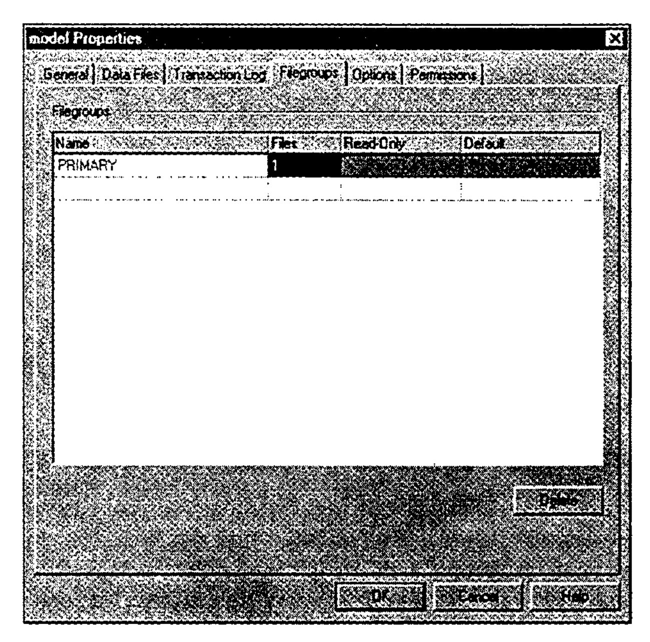


Рис. 3.8. Параметры вкладки Filegroups

Ранее уже упоминалась специальная группа файлов `PRIMARY`, к которой принадлежат все системные таблицы, а также все остальные таблицы, не принадлежащие больше ни к какой другой группе файлов. Что это дает? Путем указания принадлежности определенного файла данных группе файлов `PRIMARY` можно объединить все системные таблицы в одну группу файлов (`PRIMARY`), так что они будут физически располагаться на одном и том же носителе информации.

На заметку Внося изменения в настройки базы данных model, следует помнить, что это особая база данных, которая используется как модель для создания всех остальных баз данных. Именно поэтому к базе данных model нельзя применять те действия, которые мы вскоре применим к базе данных приложения SQLSpyNet. Дело в том, что, определив настройки базы данных SQLSpyNet в качестве настроек базы данных model, мы придем к ситуации, в которой создание новой базы данных приведет к копированию всех установок, касающихся групп файлов, что вызовет перезапись информации в уже существующих файлах данных приложения SQLSpyNet. Мягко говоря, это не слишком хорошая идея!

В целях предотвращения описанной выше ситуации SQL Server 2000 не позволяет создавать новые файлы данных, журналы транзакций или группы файлов в базе данных model, а также в любой другой системной базе данных.

При желании можно выделить в одну группу файлов все таблицы пользователей для того, чтобы разместить их на отдельном диске. А кроме этого, можно выделить в отдельную группу файлов таблицы определенной категории пользователей, отнеся к другой группе файлов таблицы всех остальных пользователей. Подобная "перетасовка" таблиц между различными группами файлов может значительно увеличить производительность базы данных в целом.

Как достичь этого на практике? Для того чтобы таблицы базы данных автоматически относились к заданной группе файлов, следует объявить ее группой файлов по умолчанию. Если же группа файлов по умолчанию не будет определена, все создаваемые таблицы будут отнесены к группе файлов PRIMARY.

Для того чтобы удалить группу файлов, необходимо выделить ее и щелкнуть на кнопке Delete (Удалить).

На заметку Прежде чем удалить заданную группу файлов, следует удалить или отнести к другой группе все содержащиеся в ней файлы данных. SQL Server 2000 не позволяет удалять группу файлов до тех пор, пока в ней еще содержатся файлы данных.

Настройка параметров базы данных model

Показанная на рис. 3.9 вкладка Options (Параметры) содержит множество различных параметров, позволяющих проводить настройку операций уровня базы данных, которые могут в значительной мере повлиять на работу приложения.

Первой группой параметров вкладки Options является Access (Доступ). Как следует из названия, параметры этой группы позволяют определить способ доступа к базе данных. Ниже приводится описание возможных способов настройки этих параметров.

- Restrict Access (Ограничить доступ) указывает SQL Server 2000 на необходимость ограничения доступа к базе данных model, разрешая его только для определенных ролей. Этот флажок устанавливается в том случае, когда необходимо запретить не входящим в указанные роли пользователям изменять настройки базы данных model. Переключатель Single User (Один пользователь) позволяет получать доступ к базе данных только одному пользователю в каждый отдельный момент времени. Не устанавливайте этот флажок — на текущем этапе разработки приложения не следует ограничивать доступ к базе данных model.
- Read-only (Только для чтения) запрещает изменение системных объектов и хранящейся в базе данных информации. До тех пор пока флажок Read-only установлен, вы не сможете внести в базу данных никаких изменений. Не устанавливайте этот флажок, так как нам необходимо, чтобы создаваемые по шаблону model базы данных можно было редактировать.

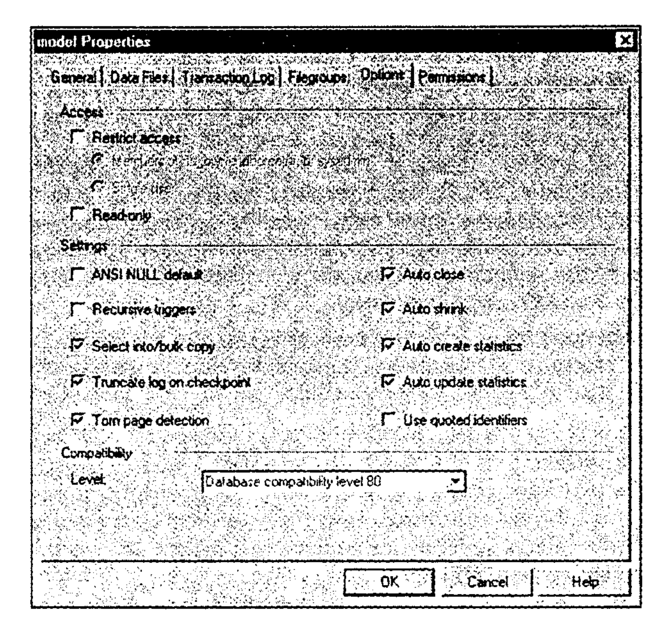


Рис. 3.9. Вкладка Options содержит параметры настройки базы данных model

Параметры второй группы вкладки Options — группы Settings (Настройки) — позволяют изменить поведение разрабатываемого приложения.

- ANSI NULL default (ANSI NULL по умолчанию) позволяет определить значение столбца таблицы, задаваемое по умолчанию (NULL или NOT NULL). В установленном состоянии этот флажок указывает на необходимость использования по умолчанию неопределенного значения NULL, что соответствует стандарту ANSI SQL-92. Это, однако, не является правилом для SQL Server 2000, так что флажок ANSI NULL default следует снять. К тому же одной из целей, ставившихся в самом начале анализа разрабатываемого приложения, было максимальное уменьшение значений NULL, что просто не позволяет упустить такую великолепную возможность. Итак, убедитесь еще раз в том, что флажок ANSI NULL default снят. В этом разделе мы еще вернемся к проблеме уменьшения неопределенных значений NULL в базе данных.
- Recursive Triggers (Рекурсивные триггеры) позволяет использовать рекурсивный вызов процедур триггеров. На текущем этапе создания приложения не будут использоваться рекурсивные триггеры, а потому этот флажок пока что не следует устанавливать.
- Select Into/Bulk Copy (Оператор SELECT INTO/Копирование большого объема данных) позволяет проводить копирование информации и структуры базы данных без занесения соответствующих сведений в журнал транзакций. Для чего это нужно? При занесении информации в базу данных сведения о проведенной транзакции записываются в журнал транзакций, что позволяет выполнить впоследствии так называемый откат (отмену) транзакции. При выполнении оператора SELECT INTO сведения о транзакции не записываются в журнал транзакций, т.е. данные скопируются намного быстрее, чем этого можно было бы ожидать в противном случае. База данных не отслеживает производимых изменений; единственным недостатком при этом является не-

возможность отката совершенной транзакции. Тем не менее нам, вероятно, придется воспользоваться этим оператором при разработке приложения, поэтому убедитесь в том, что флажок Select Into/Bulk Copy установлен.

- Truncate Log on Checkpoint (Уменьшать размер журнала транзакций в заданный момент времени) позволяет SQL Server 2000 выполнять эту операцию. Этот параметр предотвращает ситуацию, при которой размер журнала транзакций может увеличиваться экспоненциально, что, естественно, приведет в конце концов к переполнению дискового пространства. Убедитесь в том, что флажок Truncate Log on Checkpoint установлен, так как это поможет избежать некоторых нежелательных ситуаций, связанных с размером журнала транзакций.
- Torn Page Detection (Обнаружение неполных страниц) позволяет SQL Server 2000 обнаруживать неполные страницы. Сохраняемая информация записывается на диск в виде страниц. Размер страницы равен 8 Кбайт, при этом размер минимальной порции считываемых/записываемых на диск данных составляет 512 байт (эта "минимальна порция" более известна как сектор). Проведя несложный подсчет, получаем, что одна страница хранится на диске в 16 секторах. При заполнении сектора операционная система устанавливает специальный переключатель (бит, принимающий значение 0 или 1), указывающий на заполнение данного сектора. Если процесс записи информации на диск был по каким-либо причинам неожиданно прерван (например, пропало электричество), операционная система не устанавливает этот переключатель, что позволяет SQL Server 2000 обнаружить сектор, при записи информации в который произошел сбой. Другими словами, SQL Server 2000 имеет возможность обнаружить неполную страницу. Оставьте флажок Torn Page Detection установленным, так как это поможет обнаружить некорректно записанные на диск данные.
- Auto Close (Автоматическое закрытие) указывает на необходимость закрытия базы данных и завершения всех связанных с ней процессов при отключении от нее последнего пользователя. Этот параметр особенно полезен для настольных версий SQL Server 2000 (для них он является параметром, установленным по умолчанию), поскольку позволяет работать с базой данных таким образом, как если бы это был любой другой файл операционной системы, что, в частности, подразумевает возможность копирования, сжатия и пересылки файла базы данных по электронной почте. Следует отметить, что этот параметр не установлен по умолчанию для других версий SQL Server 2000, поскольку накладные расходы, связанные с автоматическим открытием и закрытием базы данных соответственно при подключении и отключении от нее пользователя, требуют больших затрат ресурсов. Если вы работаете с настольной версией SQL Server 2000, оставьте флажок Auto Close установленным, так как это поможет сберечь ресурсы вашего компьютера.
- Auto Shrink (Автоматическое уменьшение) позволяет SQL Server 2000 периодически уменьшать размер файлов данных и журналов транзакций. Уменьшение файлов происходит только в том случае, если они занимают более 25% неиспользуемого пространства. Подобно Auto Close, параметр Auto Shrink устанавливается по умолчанию только на настольных версиях SQL Server 2000. Убедитесь, что этот флажок установлен, если вы не хотите столкнуться с проблемой нехватки дискового пространства.
- Auto Create Statistics (Автоматическое создание статистики) позволяет SQL Server 2000 повысить эффективность выполнения запросов. Для этого следует проанализировать статистику выполнения запросов встроенным оптимизатором, который вычисляет наиболее быстрый способ выполнения того или

иного запроса. Auto Create Statistics является параметром, установленным по умолчанию для всех версий SQL Server 2000. При желании вы можете сбросить этот флажок и попытаться создать статистику вручную, однако гораздо легче перепоручить данную задачу SQL Server 2000. Поскольку это параметр по умолчанию, а я всегда ратую за более легкий способ выполнения работы, убедитесь в том, что флажок Auto Create Statistics установлен.

- Auto Update Statistics (Автоматическое обновление статистики) указывает SQL Server 2000 на необходимость периодического обновления статистики запросов. Данный параметр позволяет значительно повысить эффективность выполнения запросов, поскольку в этом случае оптимизатор запросов получает возможность анализа самой "свежей" информации. Как и предыдущий параметр, флажок Auto Update Statistics можно снять и создавать статистику выполнения запросов вручную. Для того чтобы максимально повысить эффективность выполнения запросов, убедитесь в том, что данный флажок установлен.
- Use Quoted Identifiers (Использовать идентификаторы, заключенные в двойные кавычки) позволяет использовать имена объектов, заключенные в двойные кавычки (" "). Как правило, данный флажок устанавливается в том случае, если имена объектов не соответствуют правилам именования Transact-SQL или содержат ключевые слова. Для того чтобы обратиться к конкретному значению (литералу) при установленном флажке Use Quoted Identifiers, следует использовать одинарные кавычки (' '). Поскольку использование ключевых слов в качестве имен (части имен) объектов не является "хорошим тоном" в программировании, флажок Use Quoted Identifiers по умолчанию снят. Тема именования объектов базы данных будет рассмотрена в ближайших разделах при создании первой таблицы разрабатываемого приложения. Тем не менее прямо сейчас стоит отметить, что, если вам не удалось избежать имен объектов, которые не соответствуют правилам именования SQL Server 2000, вы можете обратиться к ним с помощью квадратных скобок ([]). Квадратные скобки можно использовать вне зависимости от состояния флажка Use Quoted Identifiers. Не используйте некорректных имен объектов и снимите описываемый флажок. К тому же некоторые драйверы ODBC интерпретируют двойные кавычки совсем не так, как этого хотелось бы, что, естественно, является еще одним аргументом против использования параметра Use Quoted Identifiers.

На заметку Более подробно правила именования объектов и зарезервированные слова будут рассмотрены в ближайших разделах этой главы при создании первой таблицы разрабатываемого нами приложения.

И наконец, последний параметр, который можно установить для базы данных `model` (а также для всех созданных на ее основе баз данных), определяет уровень ее совместимости.

- Levels (Уровни) позволяет определить уровень совместимости базы данных с предыдущими версиями SQL Server. Основной эффект, которого можно добиться с помощью этого параметра, связан с выполнением некоторых операторов Transact-SQL и внутренней обработкой значений `NULL`, которая называется `CONCAT NULL YIELDS NULL`. Если установить совместимость базы данных на уровне SQL Server 7.0, то эта опция будет проигнорирована, так что при объединении строки со значением `NULL` в результате будет получена строка, отличная от `NULL`.

На заметку Дополнительная информация об обработке неопределенных значений NULL приведена далее в главе.

Единственная польза, которую можно извлечь из установки совместимости базы данных на уровне версии ниже 8.0 (2000), — это проведение плановых обновлений. Плановое обновление подразумевает "неспешный" переход базы данных, созданной с помощью предыдущей версии SQL Server, в формат новой версии.

Даже в том случае, когда совместимость базы данных установлена на уровне предыдущей версии, у вас остается возможность использовать все преимущества, реализованные в новой версии SQL Server, что значительно упрощает процесс планового обновления.

Поскольку разрабатываемое нами приложение создается "с нуля", убедитесь в том, что значение параметра Levels установлено в Database Compatibility Level 80.

Термин Рассмотрим составные части выражения CONCAT NULL YIELDS NULL. В данном случае CONCAT (от английского "concatenate") обозначает объединение двух строк. NULL — это упоминавшееся ранее специальное значение, которому соответствует отсутствие какого-либо значения в ячейке таблицы; следует отметить, что NULL не равняется пустой строке ("") или нулю (0). YIELDS обозначает результат выражения.

Для того чтобы более наглядно продемонстрировать смысл выражения CONCAT NULL YIELDS NULL, рассмотрим небольшой пример. Предположим, у нас есть строковая переменная со значением *MyString*. При попытке объединить ее с другой строковой переменной, имеющей значение NULL, будет получено все то же значение NULL. Формула, по которой выполняется объединение строк, выглядит примерно так: MyString+NULL=NULL.

Права доступа

Последняя вкладка диалогового окна свойств базы данных model носит название Permissions (Разрешения). С помощью параметров этой вкладки можно определить набор действий, которые разрешается проводить над базой данных конкретному пользователю или роли (роль во многом аналогична группе пользователей в Windows NT). Как следует из рис. 3.10, единственными правами, которые можно изменить для базы данных model, являются права роли Public (более подробно она рассматривается в главе 9, "Обеспечение безопасности базы данных Spy Net").

Если для пользователя или роли не будут определены права на данном уровне, то по умолчанию этот пользователь или роль и вовсе не будут иметь никаких прав. Это приведет к тому, что роль Public будет лишена права создания таблиц, представлений, хранимых процедур, не сможет провести операцию резервного копирования и т.д.

Экскурс

Так кто же является "владельцем" базы данных?

И все же почему во вкладке *Permissions* отображается одна только роль *Public*? Дело в том, что на текущем этапе не было создано ни одного нового пользователя или роли. Вопросы безопасности рассматриваются в главе 9, "Обеспечение безопасности базы данных Spy Net", а пока что приходится довольствоваться лишь одной ролью, определяемой SQL Server 2000 по умолчанию, т.е. ролью *Public*. Данную роль нельзя удалить из текущей копии SQL Server 2000, однако ее права можно изменять.

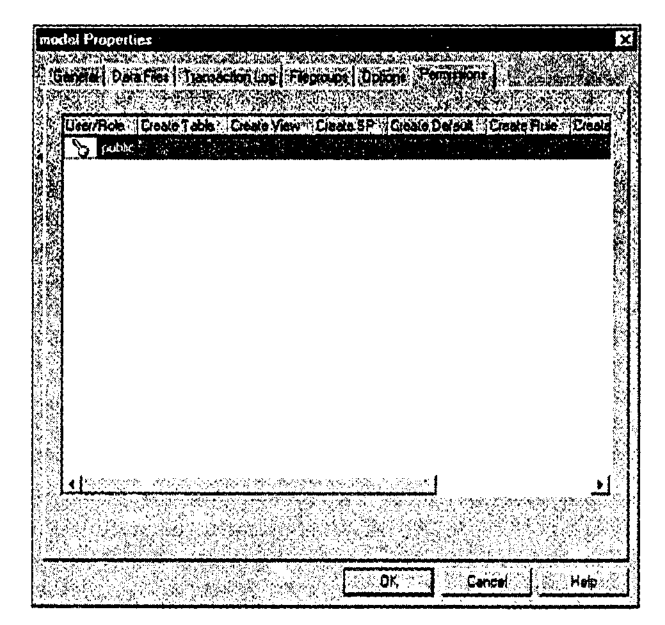


Рис. 3.10. С помощью параметров вкладки Permission можно изменить права пользователя или роли

Экскурс

Поскольку каждая база данных имеет своих пользователей, определение пользователей для базы данных model — шаг весьма опрометчивый. Причина этого очень проста: model служит шаблоном для каждой создаваемой базы данных, что подразумевает копирование всех ее объектов, включая, естественно, и определенных для базы данных model пользователей.

А как же имя пользователя sa: почему оно не отображается в списке вкладки *Permission*? Причина в том, что, хотя пользователь sa и принадлежит роли *Public*, он принадлежит еще и специальной роли *db_owner*.

Принадлежность пользователя двум различным ролям подразумевает обладание всеми правами этих ролей. Так, например, пользователь sa имеет не только права роли *Public*, но еще и права роли *db_owner*, которая является специальной системной ролью, не позволяющей изменять ее права. Последнее предотвращает ситуацию, в которой мы могли бы случайным образом лишить себя возможности управлять базой данных на уровне суперпользователя.

Что касается разрабатываемого приложения, то настройки безопасности для базы данных model лучше пока оставить на уровне заданных по умолчанию.

Причина, по которой не следует изменять настройки безопасности, заключается прежде всего в желании избежать возникновения каких-либо проблем на столь раннем этапе разработки приложения. Поскольку пользователи базы данных будут определены только в главе 9, "Обеспечение безопасности базы данных Spy Net", какое-либо изменение прав на уровне базы данных model может привести впоследствии к некорректным установкам прав доступа для отдельных пользователей.

Итак, дамы и господа, этим разделом завершается подготовительный этап, связанный с настройкой параметров баз данных, создаваемых с помощью SQL Server 2000. Следующим шагом является создание базы данных приложения SQLSpyNet, т.е., по сути, начало воплощения наших планов в действительность. Так что вставьте в накопитель CD-ROM свой любимый компакт-диск, оденьте наушники, немного расслабьтесь и — вперед, к покорению новых вершин мастерства!

Создание базы данных приложения *SQLSpyNet*

Как правило, большая часть операций в SQL Server 2000 может быть выполнена несколькими способами. Они условно делятся на две группы: способы, выполняемые с помощью графического интерфейса пользователя, и способы, подразумевающие необходимость выполнения запроса Transact-SQL непосредственно к базе данных.

Язык запросов Transact-SQL — это чрезвычайно гибкое средство управления базой данных. Стоит лишь упомянуть, что с помощью Transact-SQL можно было выполнить большую часть настроек базы данных `model`, рассмотренных в предыдущем разделе этой главы.

Так каковы же способы создания базы данных? Ниже приведены три основных способа, которые можно использовать для создания базы данных в SQL Server 2000.

- В окне Enterprise Manager запустите мастер Database Creation Wizard (Мастер создания базы данных). Это достаточно простой мастер, позволяющий выбрать основные параметры, необходимые для создания базы данных. Несмотря на то что использование мастера представляет собой весьма эффективный и не очень сложный способ создания базы данных, большинство администраторов предпочитают использовать один из двух приведенных ниже методов, что связано прежде всего с более быстрым их выполнением. Именно поэтому мы не будем рассматривать способ создания базы данных с помощью мастера Database Creation Wizard подробнее.
- Баэа данных может быть создана с помощью графического интерфейса пользователя программы Enterprise Manager. Для того чтобы создать базу данных с помощью Enterprise Manager, следует щелкнуть в дереве объектов правой кнопкой мыши над папкой Databases (Базы данных) и выбрать из контекстного меню команду New Database (Новая база данных). Благодаря дружественному интерфейсу пользователя, этот способ создания базы данных в SQL Server 2000 снискал наибольшую популярность среди всех остальных. На начальном этапе разработки приложения воспользуемся именно этим способом.
- Для того чтобы создать базу данных с помощью программы Query Analyzer, следует воспользоваться оператором Transact-SQL `CREATE DATABASE`. Несмотря на то что это один из самых сложных способов (сложность связана прежде всего с запоминанием синтаксиса оператора `CREATE DATABASE`), он позволяет достичь наибольшей гибкости, что очень ценится многими администраторами баз данных. Следует отметить, что это не единственное преимущество использования программы Query Analyzer для создания базы данных. Поскольку язык построения запросов SQL является международным стандартом, знание базового синтаксиса его операторов поможет вам при переходе на другую СУБД. (Боже упаси! И чем вам не нравится SQL Server 2000?)

Подводя итог сказанному выше, можно отметить, что SQL Server 2000 удовлетворяет двум основным требованиям, предъявляемым к нему различными категориями пользователей, — быстроте и простоте выполнения поставленной задачи.

Использование *Enterprise Manager* для создания базы данных приложения *SQLSpyNet*

Как следует из названия этого раздела, в нем рассматривается создание базы данных приложения SQLSpyNet с помощью средств графического интерфейса пользователя программы Enterprise Manager. Однако, прежде чем перейти к непосредственному созданию базы данных, следует сделать несколько оговорок, касающихся конфигурации компьютера, на котором установлен экземпляр SQL Server 2000.

По умолчанию предполагается, что такой компьютер оснащен только одним жестким диском. Естественно, что системы с двумя и более жесткими дисками не редкость, однако гораздо разумнее ориентироваться при изложении материала на так называемый "наименьший общий знаменатель", которым как раз и является система, оснащенная одним жестким диском. Следует отметить, что, если у вас есть возможность указать различные диски для файла данных и журнала транзакций, не упускайте такой шанс (причина, по которой стоит это сделать, обсуждается выше в главе).

Итак, приступим к делу!

Запустите программу Enterprise Manager и отыщите с помощью дерева объектов папку Databases. Щелкнув на ней правой кнопкой мыши, вы увидите контекстное меню, изображенное на рис. 3.11.

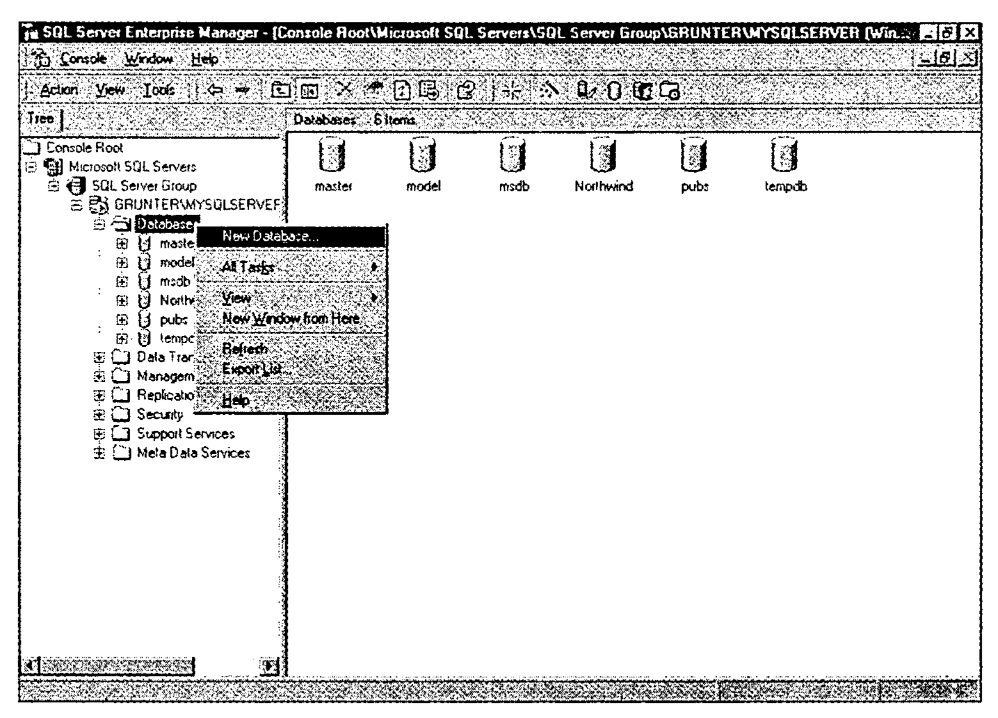


Рис. 3.11. Контекстное меню Enterprise Manager, содержащее команду создания новой базы данных

Выберите команду New Database (Новая база данных).

Появившееся в результате выполнения команды New Database диалоговое окно содержит основные настройки, которые необходимо определить при создании новой базы данных. В поле имени базы данных введите **SQLSpyNet**. Это имя будет исполь-

зоваться SQL Server 2000 для идентификации базы данных создаваемого приложения. Разумеется, вы можете назвать базу данных как-то иначе, однако следует отметить, что в оставшейся части книги я буду ссылаться на нее преимущественно по указанному выше имени. Диалоговое окно, содержащее основные параметры новой базы данных, показано на рис. 3.12.

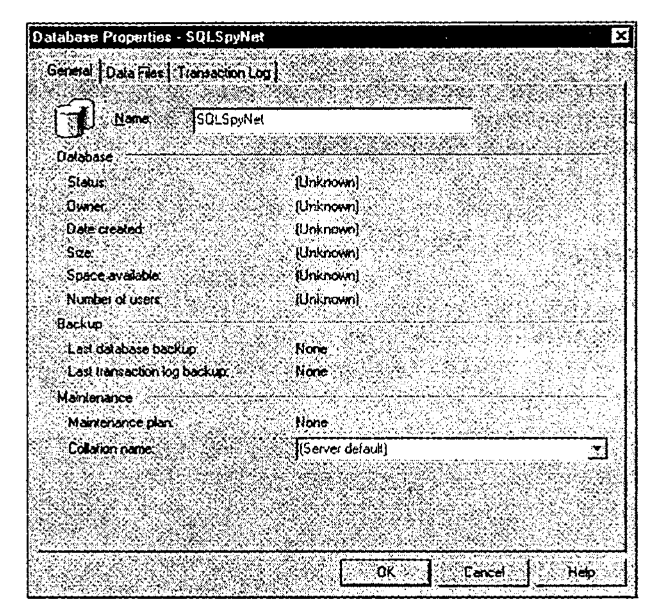


Рис. 3.12. Диалоговое окно свойств создаваемой базы данных

Вкладка *General*

Вкладка General (Общие) не отличается разнообразием предоставляемых ею опций. Помимо указания имени базы данных, единственным параметром, который здесь можно изменить, является Collation name (Имя способа представления и упорядочения символов). Данный параметр предназначен для определения национального набора символов, который используется базой данных. Единственной ситуацией, в которой может понадобиться изменить параметр Collation name, является необходимость проведения обмена информацией между базой данных и компьютером с языковыми установками, отличными от вашего компьютера.

Термин *Языковые установки* (*locale*) определяют способ отображения и хранения символов на заданном компьютере. Выбор требуемой языковой установки осуществляется на этапе инсталляции операционной системы. Разные страны используют, как правило, различные языковые установки, что позволяет отображать специфические для языка данной страны символы. Например, набору символов U.S. English соответствует языковая установка *Latin1_General*.

Поскольку разрабатываемое нами приложение создается "с нуля" и мы пока что не собираемся проводить обмен информацией с другими базами данных, следует оставить значение параметра Collation name таким, каким оно было выбрано по умолчанию при установке SQL Server 2000.

Обратите внимание на то, что в описываемом окне расположены две другие вкладки — Data Files, позволяющая изменять параметры файлов данных, и Transaction Log, позволяющая изменять параметры журнала транзакций. Параметры в этих вкладках практически полностью (за исключением имен файлов) идентичны параметрам, которые были определены для базы данных model выше в главе.

Щелкните на кнопке OK, и SQL Server 2000 создаст новую базу данных. Все гениальное просто, не правда ли?

На заметку Если только что созданная база данных не отображается в папке *Databases*, щелкните на ней правой кнопкой мыши и выберите из контекстного меню команду *Refresh* (Обновить).

Вкладка *Properties*

Выделите базу данных SQLSpyNet, щелкните на ней правой кнопкой мыши и выполните команду Properties (Свойства). На экране появится диалоговое окно, подобное изображенному на рис. 3.9. Активизируйте вкладку Options (Параметры), как показано на рис. 3.13.

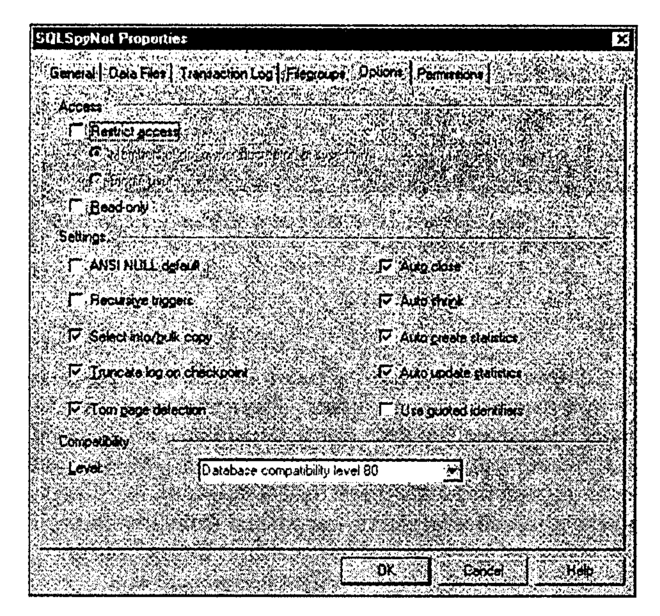


Рис. 3.13. Вкладка Options диалогового окна свойств базы данных SQLSpyNet

Посмотрите на параметры вкладки Options; они должны совпадать с аналогичными параметрами, установленными для базы данных model. Если вы удовлетворены настройкой параметров базы данных SQLSpyNet, щелкните на кнопке OK.

При более пристальном обзоре новой базы данных (особое внимание следует обратить на папки Tables (Таблицы), Views (Представления) и Stored Procedures (Хранимые процедуры)) можно отметить наличие некоторых уже созданных объектов. Это так называемые системные объекты, которые используются SQL Server 2000 для отслеживания всех остальных объектов базы данных SQLSpyNet. Например, откройте папку Tables, в которой содержится список системных таблиц.

На заметку Если, открыв папку *Tables*, вы не увидели в ней системных таблиц, измените свойства регистрации *Enterprise Manager*. Для этого выделите пиктограмму *Имя сервера\Имя экземпляра*, щелкните на ней правой кнопкой мыши и выполните команду *Edit SQL Server Registration properties* (Изменить свойства регистрации SQL Server), в результате чего на экране появится диалоговое окно свойств регистрации SQL Server.

Убедитесь в том, что флажок *Show system databases and system objects* (Отображать системные базы данных и системные объекты) выбран, после чего щелкните на кнопке *OK*.

Для того чтобы получить дополнительную информацию, касающуюся настройки SQL Server 2000, обратитесь к приложению Б, "Установка и настройка SQL Server 2000".

Системные таблицы содержат информацию, касающуюся схемы базы данных, включая имена таблиц, представлений и хранимых процедур (системная таблица sysobjects), названия, размеры и типы столбцов внутри таблиц (системная таблица syscolumns), а также имена учетных записей и права пользователей (системная таблица sysusers). Таким образом, SQL Server 2000 имеет возможность отслеживать не только таблицы, представления и хранимые процедуры, но также учетные записи пользователей и множество других объектов базы данных.

А теперь выполните процедуру, которая, с одной стороны, может показаться вам несколько странной, а с другой — необходимой на текущем этапе разработки приложения. Речь идет об удалении только что созданной базы данных.

Как вы наверняка помните, существует несколько способов создания базы данных в SQL Server 2000. Пока что изучен только один из них — применение графического интерфейса пользователя программы Enterprise Manager. Теперь на очереди один из наиболее гибких методов создания базы данных, основанный на использовании кода Transact-SQL.

На заметку В оставшейся части книги будет использоваться "смесь" двух способов управления базой данных — кода Transact-SQL и графического интерфейса пользователя *Enterprise Manager*. Следует отметить, что выполнение дважды одной и той же задачи с использованием для этого различных средств отнюдь не входит в наши планы. Вместо этого, выполнив одну задачу (с помощью Transact-SQL или *Enterprise Manager*), мы сразу же сосредоточимся на следующей, так как в противном случае нам вряд ли удастся быстро и качественно завершить разработку приложения SQLSpyNet.

Удаление базы данных с помощью кода Transact-SQL

Прежде чем заново создать базу данных SQLSpyNet с помощью кода Transact-SQL, необходимо избавиться от ее предшественницы, созданной с помощью графического интерфейса Enterprise Manager.

Запустите программу Query Analyzer, воспользовавшись для этого меню Start (Пуск) или выбрав команду Tools⇨SQL Query Analyzer (Средства⇨SQL Query Analyzer) из строки меню приложения Enterprise Manager. В главном окне программы Query Analyzer введите код Transact-SQL, приведенный в листинге 3.1, как показано на рис. 3.14.

Рис. 3.14. Удаление базы данных с помощью кода Transact-SQL

Листинг 3.1. Удаление базы данных SQLSpyNet с помощью кода Transact-SQL

Код для запуска

```
USE master
GO
IF EXISTS (SELECT name FROM master.dbo.sysdatabases
WHERE name = 'SQLSpyNet')
DROP DATABASE [SQLSpyNet]
GO
```

Введя код, воспользуйтесь пиктограммой с изображением зеленого треугольника или комбинацией клавиш <Alt+x> для того, чтобы его выполнить.

В результате выполнения кода вы должны увидеть два сообщения, примерный текст которых приведен ниже.

```
1: Deleting database file
    ↳'d:\program files\windows nt\Microsoft SQL
    ↳Server\MSSQL$MYSQLSERVER\data\SQLSpyNet_Log.LDF'.
2: Deleting database file
    ↳'d:\program files\windows nt\Microsoft SQL
    ↳Server\MSSQL$MYSQLSERVER\data\SQLSpyNet_Data.MDF'.
```



Сохраните путь к файлу данных и журналу транзакций, так как он понадобится при повторном создании базы данных SQLSpyNet.

Эти сообщения подтверждают факт удаления файла данных и журнала транзакций базы данных SQLSpyNet.

На заметку Если вместо *SQLSpyNet* вы использовали какое-то другое имя базы данных, замените им все вхождения имени *SQLSpyNet* в приведенном выше коде Transact-SQL.

Анализ Смысл приведенного в листинге 3.1 кода достаточно прост. В строках 1 и 2 указывается необходимость использования базы данных master для выполнения последующих операторов Transact-SQL. Следует отметить, что все операции уровня базы данных, такие, как CREATE (создание) или DROP (удаление), должны выполняться непосредственно из базы данных master.

В строках 3 и 3а осуществляется проверка наличия базы данных SQLSpyNet в пределах заданного экземпляра SQL Server 2000. Используемый для этого оператор Transact-SQL называется оператором ветвления кода; более подробно он рассматривается в главе 4, "Обработка данных с помощью кода Transact-SQL".

Четвертая строка кода содержит оператор удаления базы данных, а в пятой указывается необходимость выполнения двух предыдущих операторов Transact-SQL.

Не волнуйтесь, если вы пока что не понимаете смысла операторов `SELECT` или `WHERE`; он подробно разъясняется в главе 4.

Обновив содержимое папки Databases с помощью средств Enterprise Manager или Query Analyzer (используя Object Browser, обсуждаемый более подробно в главе 2, "Компоненты SQL Server 2000"), вы обнаружите, что SQLSpyNet исчезла из списка баз данных, существующих в пределах заданной копии SQL Server 2000.

Итак, на текущий момент мы уже успели создать и удалить базу данных SQLSpyNet с использованием для этого двух различных средств SQL Server 2000. Следующим этапом является воссоздание SQLSpyNet с помощью Query Analyzer, так что вы правильно сделали, если еще не закрыли окно этого приложения.

Использование языка определения данных (DDL) для создания базы данных и ее объектов

С использованием новых возможностей Query Analyzer создание общей схемы наиболее распространенных объектов базы данных превращается в довольно простое задание. С помощью нескольких щелчков мышью и небольшого фрагмента кода можно очень быстро создать новую базу данных.

В этом разделе гораздо подробнее рассматривается структура оператора `CREATE DATABASE`, нежели использование предоставляемых SQL Server 2000 стандартных шаблонов кода.

Однако, прежде чем рассматривать способ создания базы данных с помощью оператора `CREATE DATABASE`, необходимо прояснить несколько важных моментов, касающихся языка SQL. Как упоминалось ранее, SQL является стандартным языком, предназначенным для определения реляционных баз данных и доступа к ним. Математической основой этого языка выступает теория множеств. SQL условно делится на две части — язык определения данных и язык управления данными.

Термин *Язык определения данных (Data Definition Language — DDL)* используется для создания объектов и определения способа представления информации в базе данных. С помощью языка определения данных можно не только создавать базы данных, таблицы, представления и т.д., но и удалять их (как раз то, что было продемонстрировано в предыдущем примере). Язык определения данных можно легко отличить по используемым в нем операторам языка SQL, которые, как правило, начинаются словами `CREATE`, `ALTER` или `DROP`.

Термин *Язык управления данными (Data Manipulation Language — DML)* используется для проведения различных операций над хранящейся в базе данных информацией. С помощью языка управления данными можно не только вносить в базу данных новые строки информации, но и удалять их. Язык управления данными можно легко отличить по используемым в нем операторам языка SQL, которые, как правило, начинаются словами `SELECT`, `INSERT`, `UPDATE` или `DELETE`.

Учитывая сказанное и просмотрев еще раз код Transact-SQL, который использовался для удаления базы данных, можно легко сделать вывод, что этот код представляет собой смесь языка определения данных и языка управления данными. Если говорить более конкретно, то оператор `DROP`, расположенный в строке 4, является типичным примером DDL, в то время как `SELECT` из строки 3 представляет собой оператор DML.

Получив основное представление о языках DDL и DML, приступим к повторному созданию базы данных SQLSpyNet, используя для этого оператор DDL `CREATE DATABASE`.

`CREATE DATABASE` является довольно сложным оператором, что прежде всего связано с его синтаксисом, позволяющим определить большое число параметров. Учитывая ограниченный объем книги, рассмотрим только те параметры, которые имеют наибольшее значение для разработки приложения. В листинге 3.2 приведен код Transact-SQL, который используется для повторного создания базы данных SQLSpyNet.

Листинг 3.2. Создание базы данных SQLSpyNet с помощью кода Transact-SQL

Код для запуска

```
1:  USE master
2:  GO
3:  CREATE DATABASE SQLSpyNet
3a: ON PRIMARY (NAME = 'SQLSpyNet_Data',
3b: FILENAME =
3c: 'x:\SQLSpyNet_Data.MDF',
3d: SIZE = 5MB,
3e: FILEGROWTH = 10%)
4:  LOG ON (NAME = 'SQLSpyNet_Log',
4a: FILENAME =
4b: 'x:\SQLSpyNet_Log.LDF',
4c: SIZE = 5MB,
4d: FILEGROWTH = 10%)
5:  GO
```

Обратите внимание на то, что все вхождения подстроки *x:* должны быть заменены путями к файлу данных и журналу транзакций.



В качестве путей к файлу данных и журналу транзакций можно указать пути, использовавшиеся при первом создании базы данных SQLSpyNet. Если вы не сохранили их, введите вместо x:\ новые пути.

Указав пути к файлу данных и журналу транзакций, выполните приведенный выше код Transact-SQL. В качестве результата SQL Server 2000 возвратит две строки:

```
1: The CREATE DATABASE process is allocating 5.00 MB
   ↳on disk 'SQLSpyNet_Data'.
2: The CREATE DATABASE process is allocating 5.00 MB
   ↳on disk 'SQLSpyNet_Log'.
```

Проанализируем код Transact-SQL, приведенный в листинге 3.2.

Анализ

- В строке 1 указывается необходимость использования при выполнении всех последующих операторов Transact-SQL базы данных master.
- В строке 2 находится команда, приводящая к выполнению кода в строке 1. Команда GO используется в тех случаях, когда необходимо незамедлительно выполнить какой-либо фрагмент кода Transact-SQL, не дожидаясь выполнения всего блока. Команда GO является одной из особенностей всех версий SQL Server. В некоторых других СУБД для незамедлительного выполнения блока кода используется символ точки с запятой (;). Путем установки различных параметров можно переопределить способ, благодаря которому SQL Server 2000 будет воспринимать необходимость выполнения фрагмента кода; однако на данном этапе это излишне, так как пока что код других СУБД использоваться не будет.
- В строке 3 находится оператор DDL, создающий базу данных. Единственным параметром, который передается этому оператору, является ее имя. К сожалению, такой способ создания базы данных не предоставляет нам какой-либо гибкости.
- В строке 3а указывается используемая группа файлов (по умолчанию — PRIMARY), а также имя файла данных. Открывающая круглая скобка обозначает начало определения параметров файла данных. Параметр NAME = позволяет задать имя, которое будет использоваться SQL Server 2000 при обращении к файлу данных.
- В строках 3b и 3c определяется местоположение файла данных.
- В строках 3d и 3e указывается начальный размер файла данных, а также параметр его приращения. По умолчанию размер приращения файла определяется в мегабайтах, так что не забудьте поставить знак %, если хотите определить величину приращения файла в процентах от его размера. Закрывающая круглая скобка обозначает окончание определения параметров файла данных.
- В строке 4 указываются параметры журнала транзакций. Открывающая круглая скобка обозначает начало определения параметров. Параметр NAME = позволяет задать имя, которое будет использоваться SQL Server 2000 при обращении к журналу транзакций.
- Определяемые в строках 4a–4d параметры журнала транзакций полностью аналогичны соответствующим параметрам файла данных (см. выше).

Анализ

- В строке 5 указывается необходимость выполнения предыдущего блока кода Transact-SQL (начиная от последнего вызова команды GO). В данном случае этот блок охватывает все строки кода, начиная со строки с оператором CREATE DATABASE.

Обновив содержимое папки Databases, используя для этого дерево объектов Enterprise Manager или окно Object Browser программы Query Analyzer, вы увидите вновь созданную базу данных SQLSpyNet. Как и в случае с Enterprise Manager, создание базы данных с помощью кода Transact-SQL и программы Query Analyzer не выглядит чересчур сложным.

Убедитесь в том, что вновь созданная база данных удовлетворяет всем ранее сформулированным требованиям (для этого откройте диалоговое окно свойств базы данных SQLSpyNet). Если вы найдете отличия, устраните их.

Как упоминалось ранее, при создании базы данных с помощью кода Transact-SQL можно определить много различных параметров. Например, можно запретить удаление информации из базы данных, указать требуемую языковую настройку или группу файлов. Подводя итог, следует отметить, что разработка базы данных с помощью кода Transact-SQL несет в себе гораздо больше гибкости, однако, с другой стороны, она намного сложнее разработки базы данных с помощью графического интерфейса пользователя программы Enterprise Manager.

В следующем разделе проводится краткий обзор таблиц базы данных SQLSpyNet и рассматриваются некоторые концепции теории баз данных.

Краткий обзор таблиц базы данных SQLSpyNet

Таблица является основным элементом любой базы данных. С помощью таблиц можно хранить и извлекать логически связанную информацию. Лишившись таблиц, мы не смогли бы сохранить в базе данных ни бита информации, что попросту привело бы к вырождению самого понятия "база данных".

В таблицы можно вносить не только такую информацию, как имена тайных агентов, адреса проживания, даты рождения и т.д., но и более абстрактные понятия, например деятельность, в которую вовлечены тайные агенты и злоумышленники.

Таблица может быть связана с другими таблицами, не связана ни с одной таблицей или использоваться в качестве временного хранилища информации. Хорошо спроектированная таблица гарантирует, что хранящаяся в ней информация организована наилучшим образом, т.е. точно и лаконично представлена в базе данных.

В SQL Server 2000 используются три основных типа таблиц, краткие описания которых приводятся ниже.

- *Пользовательские таблицы* (*user-defined tables*). Создаются разработчиками приложений, которым нужно хранить в базе данных определяемую конкретной предметной областью информацию (например, данные о тайных агентах и злоумышленниках). Пользовательские таблицы легко обнаружить с помощью дерева объектов Enterprise Manager, поскольку их тип определен как *User* (Пользователь). В окне Object Browser программы Query Analyzer пользовательские таблицы находятся в папке User, вложенной в папку Tables (Таблицы).
- *Системные таблицы* (*system tables*). Используются внутренним механизмом SQL Server 2000 для управления объектами базы данных. Согласно принятым правилам именования, названия всех системных таблиц начинаются на *sys*, например `sysobjects` (эта таблица содержит список всех объектов базы данных).
- *Временные таблицы* (*temporary tables*). Используются в качестве временного хранилища информации. Согласно принятым правилам, имена всех времен-

ных таблиц начинаются символом хэша (#). Различные типы временных таблиц имеют разные уровни видимости.

Более подробно каждый из типов таблиц рассматривается в ходе дальнейшей разработки приложения SQLSpyNet.

Так почему же для хранения информации в базе данных используются именно таблицы? Дело в том, что таблицы моделируют объекты реального мира (например, людей, тайных агентов, злоумышленников и т.д.) и события, или отношения, существующие между этими объектами (например, деятельность, в которую вовлечены тайные агенты и злоумышленники). Более подробно связь между объектами реального мира и таблицами прослеживается с помощью диаграммы отношений между объектами, которая была спроектирована в главе 1, "SQLSpyNet: от идеи до воплощения в виде базы данных SQL Server 2000".

Следует отметить, что таблица, не связанная ни с какой другой таблицей, дает очень мало преимуществ при использовании. В конце концов, с тем же успехом можно организовать всю необходимую информацию в виде одного-единственного файла! Организация отношений между таблицами является ключом к созданию эффективного и информативного хранилища данных.

Отношения между таблицами могут поразительно точно копировать отношения между объектами реального мира. Например, в реальном мире деятельность тайного агента направлена на борьбу со злоумышленником и с его планами обрести мировое господство. В базе данных SQLSpyNet этому отношению соответствует ассоциативная таблица, в которой хранится информация о результате борьбы между тайным агентом и злоумышленником.

На заметку Естественно, что выбранная предметная область напоминает скорее захватывающий шпионский комикс, нежели реально существующую задачу (если только заказчиком такого приложения не является главное управление внешней разведки). Тем не менее следует отметить, что все обсуждаемые здесь вопросы идеальным образом переносятся в область деловых отношений, например между продавцом и покупателем, которые характеризуются датой и количеством проданного товара, адресом проживания покупателя (если ему требуется доставка на дом) и т.д. К тому же стоит всего лишь представить, что большую часть карьеры вам придется посвятить бесконечным записям о продажах товаров, как вы сразу же поймете, что перед вами изумительный шанс изучить SQL Server 2000 на примере чрезвычайно интригующего задания. В конце концов, только вам решать, кто в этом мире может оказаться злоумышленником.

Помимо всего прочего, отношения между таблицами помогают в значительной степени повысить целостность хранящейся в базе данных информации. Определив отношение между двумя таблицами с помощью первичного и внешнего ключей, вы тем самым устанавливаете некое правило, в соответствии с которым каждой записи в таблице с внешним ключом (назовем ее таблицей-потомком) должна соответствовать запись с таким же значением первичного ключа в таблице-родителе.

В контексте базы данных SQLSpyNet иллюстрацией сказанного является невозможность добавить запись в таблицу тайных агентов (`Spy`), если ей не соответствует запись в таблице людей (`Person`).

Возврат к анализу структуры приложения SQLSpyNet

Возвращаясь к анализу структуры приложения SQLSpyNet, следует вспомнить, что в главе 1, "SQLSpyNet: от идеи до воплощения в виде базы данных SQL

Server 2000", было принято решение о реализации таблиц тайных агентов (Spy) и злоумышленников (BadGuy) в виде подтипов таблицы Person. Помимо этого, таблица Activity определена как ассоциативная между таблицами Spy и BadGuy.

Что касается оставшейся части таблиц, то здесь следует напомнить, что данная книга является не справочником по проектированию баз данных, а руководством по SQL Server 2000, так что полный вариант диаграммы отношений между объектами будет представлен без детальных (и, по большому счету, скучных) объяснений. Тем не менее в данном разделе все-таки приводится поверхностное описание полной версии изображенной на рис. 3.15 диаграммы отношений между объектами для того, чтобы дать читателю хоть какое-то представление о структуре базы данных SQLSpyNet.

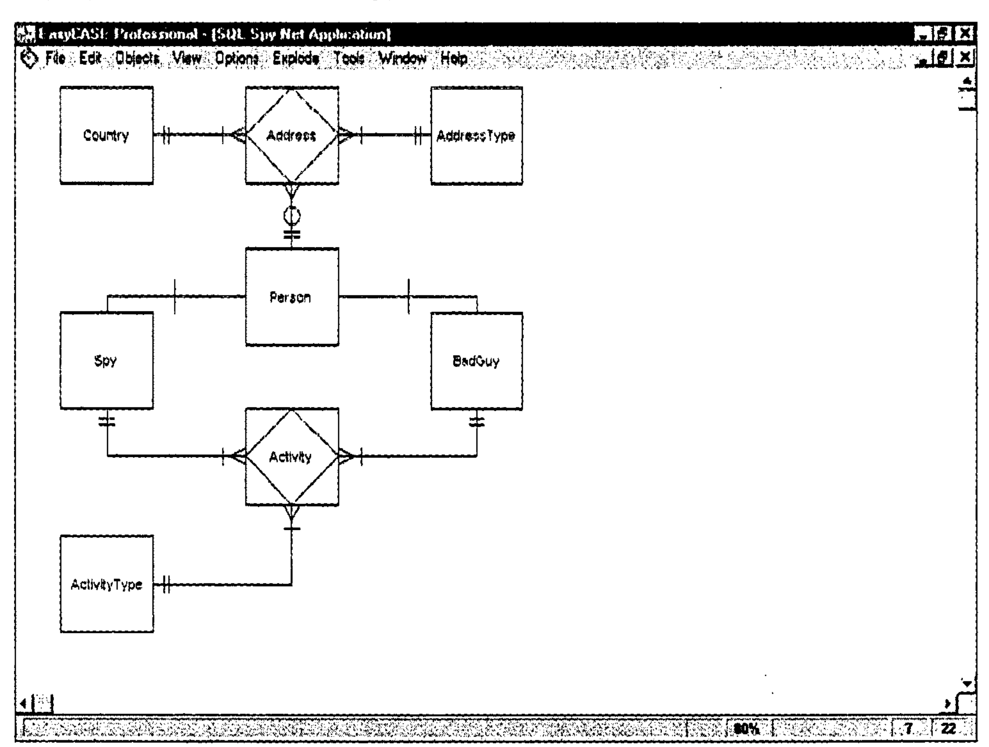


Рис. 3.15. Окончательная версия диаграммы отношений между объектами базы данных SQLSpyNet

Как видите, диаграмма отношений между объектами для всего приложения получилась не очень-то громоздкой. Она содержит всего лишь несколько таблиц. Обратите внимание на отношение типа "родитель–потомок" между таблицами Spy, BadGuy и таблицей Person.

Как отмечалось в главе 1, "SQLSpyNet: от идеи до воплощения в виде базы данных SQL Server 2000", такая схема организации отношений между таблицами позволяет добиться большой гибкости приложения в смысле дальнейшего расширения его структуры. Например, при необходимости добавить в базу данных таблицу служащих следует всего лишь создать еще один подтип таблицы Person.

Особого внимания заслуживает способ отношения между таблицами Spy и BadGuy. Тайный агент получает свое задание лишь в том случае, если какой-нибудь зло-

умышленник в очередной раз пытается осуществить зловещий план достижения мирового господства. Именно этот принцип и положен в основу отношения между таблицами Spy и BadGuy.

Поскольку в реальной жизни тайный агент может пытаться разрушить один или сразу несколько планов злоумышленников, которые, в свою очередь, могут иметь несколько коварных планов "про запас", тип отношения между таблицами Spy и BadGuy был определен как "многие ко многим".

Напомним, что связь между таблицами типа "многие ко многим" не может быть непосредственно реализована в рамках реляционной базы данных, при этом платить за такую ограниченность приходится использованием дополнительной (ассоциативной) таблицы. Введенная в рамках приложения SQLSpyNet ассоциативная таблица называется Activity. В ней хранится информация о злоумышленниках, с которыми приходится иметь дело тайному агенту, а также те зловещие планы, которые тайный агент уже успел разрушить.

В таблице ActivityType содержится список коварных планов, вынашиваемых злоумышленниками. Как правило, эти люди не отличаются особой оригинальностью, поэтому наиболее распространенными являются планы, в основе которых лежит желание обрести мировое господство. Более "миролюбивыми" в этом смысле являются планы по нарушению гражданского спокойствия и сеянию паники среди людей.

Таблица Address используется для хранения различных адресов человека. Поскольку один человек может иметь сразу несколько адресов, эта информация выделена в отдельную таблицу, что позволило, во-первых, уменьшить повторяющиеся строки в таблице Person, а во-вторых, привести нашу модель данных (диаграмму отношений между таблицами) в соответствие с 1НФ.

Термин

1НФ (*1NF*) — это сокращение от "первая нормальная форма" (First Normal Form), которая является первым уровнем нормализации (см. экскурс ниже). Определяемое первой нормальной формой правило заключается, по существу, в запрете хранения в ячейке таблицы списка значений (точнее, разделенного запятыми списка значений), а также в запрете повторяющихся групп.

Экскурс

1НФ, 2НФ, 3НФ, 4!

Истоки термина *нормализация* следует искать в реляционной теории, где это понятие было впервые сформулировано и предложено отцом реляционной теории Е.Ф. Коддом. Нормализация — это процесс организации таблиц и хранящихся в них данных таким образом, чтобы максимально уменьшить число лишних зависимостей и неэффективных структур.

Нормализация линейна по своей сути (в том смысле, что каждое последующее правило может быть применено лишь после выполнения всех предыдущих правил); она используется для определения наиболее оптимальной структуры базы данных, которая возможна в заданных условиях.

В процессе нормализации определяются шесть нормальных форм, или этапов. Как было сказано выше, правила нормализации линейны по своей сути; например, второе правило зависит от того, было ли выполнено первое, третье зависит от второго и т.д.

Излюбленной большинством разработчиков баз данных нормальной формой является 3НФ (3NF). Как правило, к этой форме приводится большая часть всех существующих баз данных. Третья нормальная форма пред-

ставляет собой оптимальный компромисс между двумя противоположными тенденциями — стремлением к нормализации и необходимостью сохранить функциональные возможности и простоту реализации. К тому же большинство разработчиков утверждают, что ЗНФ вполне достаточно для обеспечения должного уровня непротиворечивости данных.

На заметку Удалив повторяющиеся строки из таблицы `Person`, ее можно привести к первой нормальной форме (1НФ).

Выше уже упоминалось, что определение модели данных — субъективный процесс; учитывая это, следует отметить, что предложенная модель базы данных приложения SQLSpyNet может не понравиться некоторым читателям. Наиболее вероятная критика, которая обрушится с их стороны на рассмотренную выше диаграмму отношений между объектами, будет связана с ее ограничениями. Дабы упредить всевозможные несправедливые нападки, ниже приводится список ограничений, которые (увы!) имеет предложенная мною модель данных приложения SQLSpyNet.

- В каждый отдельно взятый момент времени тайный агент может находиться только на одном-единственном задании. Таким образом, исключается возможность возникновения ситуации, когда один тайный агент противостоит зловещим планам двух или более злоумышленникам (что, кстати вполне реально). Аналогично, злоумышленник не может одновременно вынашивать несколько планов по достижению мирового господства (или, на худой конец, по захвату чьих-то носков).
- Любой человек может иметь несколько различных адресов, однако один адрес может соответствовать только одному человеку. В общем, можете забыть о коммуналках и общежитиях! Для того чтобы обеспечить соответствие одного адреса нескольким разным людям, следует либо ввести новую таблицу (правильный путь), либо разрешить повторение строк в таблице `Address` (очень дурной тон, лучше побыстрее забудьте о такой возможности).
- И наконец, хотя это и выглядит наиболее невероятным, можно прийти к такой ситуации (а мы к ней и придем, если только не сможем четко определить некоторые правила), когда один человек одновременно будет и тайным агентом, и злоумышленником. Чтобы избежать этого, необходимо ввести *ограничения домена*, которые помогут справиться с проблемой ввода некорректной информации в ту или иную таблицу базы данных.

Термин *Ограничения домена* (*domain constraints*) позволяют наложить определенные правила на вводимые в столбец таблицы данные. Например, если вводимое значение находится в промежутке от 1 до 10, оно будет принято базой данных, если нет — ввод такого значения будет запрещен. Стоит ли говорить, что ограничения домена представляют собой одно из наиболее мощных средств, доступных разработчику базы данных.

Итак, на данном этапе практически полностью завершен процесс формирования диаграммы отношений между объектами приложения SQLSpyNet. Слово “практически” говорит о том, что еще не выбраны имена для столбцов каждой таблицы, однако это лучше делать непосредственно на этапе их создания.

Воплощение созданной диаграммы отношений между объектами в базе данных SQLSpyNet

Несмотря на то что в данной главе уже представлен достаточно обширный материал, только сейчас разработка приложения SQLSpyNet начинает идти "полным ходом". С одной стороны, есть база данных, с другой — модель отношения между ее объектами; таким образом, осталось лишь воплотить модель отношения между объектами в базе данных SQLSpyNet.

Воплощение диаграммы отношений между объектами в базе данных SQLSpyNet — довольно простая, можно даже сказать, прямолинейная задача. При ее решении будем использовать как приложение Enterprise Manager, так и Query Analyzer.

В оставшейся части главы рассматриваются некоторые наиболее распространенные наработки в области создания приложений, использующих базу данных. Следует отметить, что, хотя эти наработки признаются многими организациями во всем мире, существуют и другие подходы, которые все еще принимаются на вооружение отдельными организациями.

Единственно неоспоримый тезис, касающийся всевозможных стандартов, утверждает, что стандарты практически постоянно улучшаются и перерабатываются. Приступая к воплощению диаграммы отношений между объектами в базе данных SQLSpyNet, познакомимся с одним из наиболее распространенных стандартов — соглашением относительно способа именования объектов.

Создание первой таблицы базы данных приложения *SQLSpyNet*

Итак, несмотря на мою болтовню, мы наконец-то вплотную подошли к созданию первой таблицы базы данных приложения SQLSpyNet!

В процессе создания таблиц базы данных SQLSpyNet мы будем часто обращаться к диаграмме отношений между объектами с тем, чтобы заполнить существующие там пробелы (которые в основном касаются имен полей таблиц).

На заметку

Как правило, разработка приложения на уровне программного кода начинается только после полного завершения фазы его начального проектирования (за исключением, может быть, некоторых несущественных деталей). Именно поэтому, предугадывая ваш страстный порыв начать все сразу, здесь и сейчас, я чувствую себя обязанным предостеречь вас от некоторых подводных камней.

Так почему же не рекомендуется создавать программный код любого приложения до завершения фазы его проектирования? Ответ прост: такая практика, вероятнее всего, приведет к необходимости переделывать впоследствии большую часть работы. Что касается разработки базы данных, то здесь типична ситуация, когда слишком рано созданная таблица постепенно все больше и больше "не вписывается" в структуру всего приложения, что зачастую требует ее уничтожения и повторного создания. А если учесть, что сроков и бюджета разработки приложения всегда чуть-чуть не хватает, то трата впустую половины времени и денег оказывается крайне непозволительной роскошью.

На заметку И хотя, честно говоря, при разработке приложения SQLSpyNet указанное выше "золотое правило" соблюдается не в полной мере, я чувствую, что если мы не начнем прямо сейчас создавать первую таблицу, то вы вряд ли захотите продолжить чтение этой книги.

Итак, приступим.

Выделите базу данных SQLSpyNet в дереве объектов программы Enterprise Manager и щелкните на папке Tables (Таблицы). Вы увидите список таблиц, существующих на текущий момент в базе данных (в нашем случае это будут только системные таблицы).

Щелкните правой кнопкой мыши над папкой Tables, в результате чего откроется контекстное меню с командой создания новой таблицы (рис. 3.16).

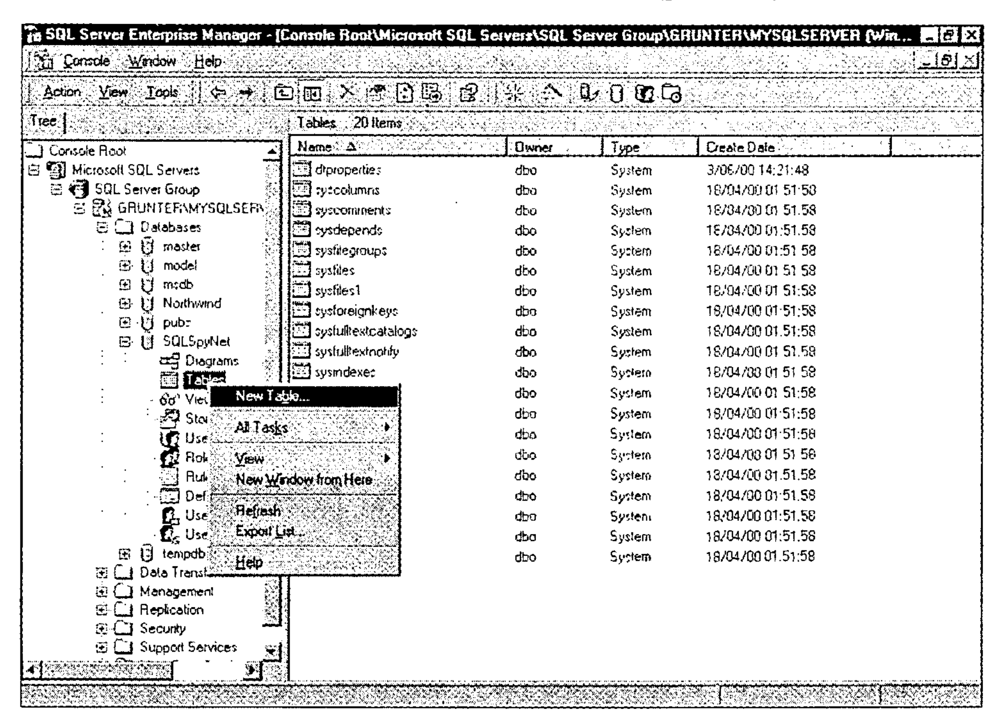


Рис. 3.16. Команда создания новой таблицы базы данных SQLSpyNet

В результате выполнения команды создания таблицы открывается новое диалоговое окно. С первого взгляда можно отметить несомненное сходство этого окна с окном редактора таблиц программы Microsoft Access, что также является еще одним из нововведений SQL Server 2000. С помощью редактора таблиц SQL Server 2000 можно создавать столбцы, первичные ключи, индексы, а также снабжать столбцы подробным описанием (еще одно новое свойство SQL Server 2000).

Воплощая диаграмму отношений между объектами в базе данных, следует всегда начинать с таблиц самого верхнего уровня, поскольку только так можно гарантировать доступность всех необходимых таблиц на этапе определения отношений между ними. В данном случае, например, следует начать с таблицы `Person`, поскольку она не содержит ни одного внешнего ключа, связывающего ее с другими таблицами. Необходимо отметить, что подобным таблице `Person` свойством обладают также таблицы `ActivityType` и `AddressType`.

На рис. 3.17 изображен редактор таблиц, в котором определены все столбцы таблицы `Person`, а также ее первичный ключ.

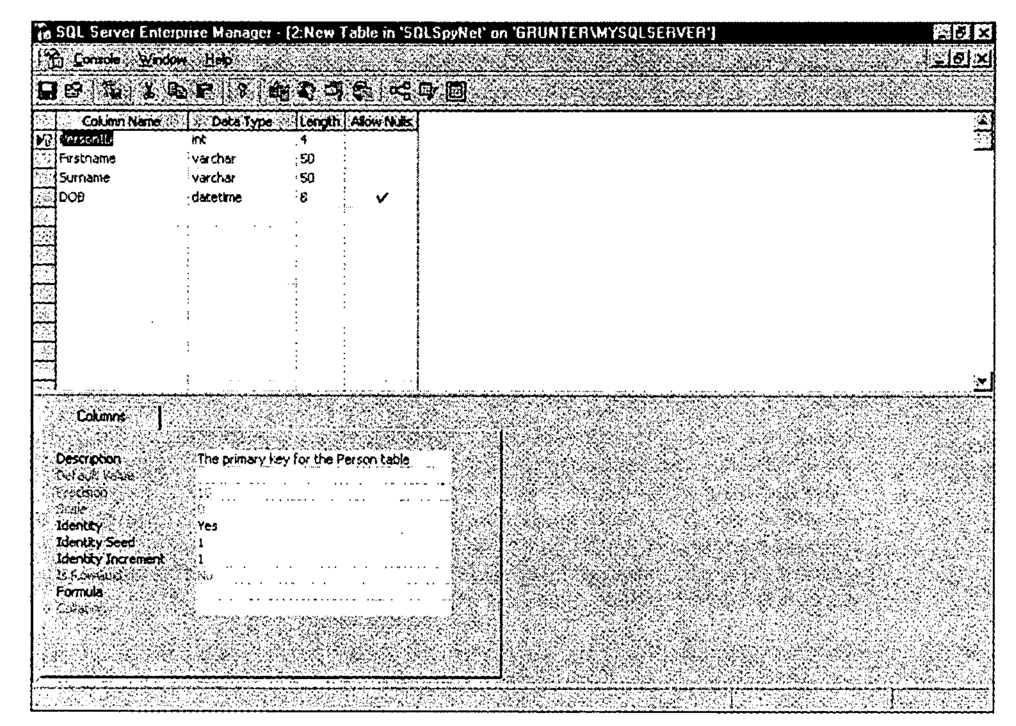


Рис. 3.17. Столбцы таблицы `Person`

Идентификационным столбцом таблицы `Person` является `PersonID`. При внесении в таблицу каждой новой строки значение идентификационного столбца автоматически увеличивается. Например, если его значение было равно 1, то при внесении в таблицу новой строки SQL Server 2000 увеличит его до 2 (это справедливо только в том случае, если автоматическое приращение значения идентификационного столбца равняется 1, что, кстати, является установкой по умолчанию). Каждое последующее внесение в таблицу новой строки вызовет соответствующее увеличение значения идентификационного столбца. Следует отметить, что удаление строки из таблицы никоим образом не влияет на значения, хранящиеся в идентификационном столбце.

Экскурс

Так как же нам назвать наши столбцы?

Прежде чем окончательно решиться дать то или иное имя столбцу таблицы, следует принять во внимание изложенное ниже.

SQL Server 2000 налагает определенные ограничения на символы, которые можно использовать в именах столбцов таблиц. Первым в имени столбца может быть символ из наборов `a-z`, `A-Z`, а также символы `_`, `@` или `#`. Все остальные символы в имени столбца можно выбирать из упомянутых выше, а кроме того, использовать цифры и символ `$`. Минимальная длина имени столбца равняется одному символу, а максимальная — 128. Несмотря на то что SQL Server 2000 допускает использование пробела в имени столбца, такая практика категорически не рекомендуется, поскольку в этом случае при обращении к данному столбцу приходится использовать квадратные скобки (`[]`).

Определим структуру таблицы `Person`, следуя приведенным ниже указаниям. Создадим столбцы и назначим один из них первичным ключом. Следует отметить, что, кроме ограничений, упомянутых в приведенном выше экскурсе,

SQL Server 2000 накладывает на имена столбцов некоторые дополнительные ограничения. Все дело в том, что SQL Server 2000 имеет список так называемых *зарезервированных слов*, которые не должны использоваться в качестве имен столбцов; список зарезервированных слов можно найти в справке по SQL Server 2000.

Термин

Зарезервированное слово (*reserved word*) — это специальный термин, который используется SQL Server 2000 для описания некоторого объекта (в том числе функции, процедуры и т.д.), имеющего специальное значение. Одним из таких слов, например, является зарезервированное слово CREATE. Правило, налагающее запрет на использование зарезервированных слов в SQL Server 2000, аналогично правилам, запрещающим использование зарезервированных слов в таких языках программирования, как Visual Basic или C++.

На заметку

Несмотря на сказанное выше, в качестве имен столбцов все-таки можно использовать зарезервированные слова, заключив их предварительно в квадратные скобки. Так, в синтаксическом аспекте [SELECT] представляет собой вполне допустимое имя. Следует помнить, что подобный стиль именования столбцов может в конечном итоге привести к возникновению ошибки, так как некоторые драйверы ODBC все еще не вполне корректно интерпретируют квадратные скобки в именах столбцов.

- Прежде всего убедитесь в том, что флажок Allow Nulls (Разрешить значения NULL) для столбца PersonID снят. Необходимо помнить, что идентификационный столбец, которым как раз и является PersonID, ни в коем случае не должен хранить неопределенные значения NULL. Для того чтобы получить дополнительную информацию о значениях NULL, которая во многом объясняет мое стремление во что бы то ни стало от них избавиться, читайте материал следующего экскурса.

Экскурс

Уменьшение количества значений NULL в приложении SQLSpyNet

Несмотря на то что проблема появления в приложении значений NULL обсуждалась ранее в этой книге, только для того, чтобы освежить вашу память, я напомню, что NULL — это неопределенное значение. Значение NULL не равно нулю (0) или пустой строке (""). Более того, NULL не равно другому такому же значению NULL! При попытке сравнения значения NULL с любым другим значением в качестве результата будет возвращено опять-таки значение NULL (последнее утверждение зависит от настроек СУБД).

Существует много различных мнений, касающихся роли значений NULL в базе данных, и я наверняка разочарую некоторых читателей, заявив, что мне, мягко говоря, не очень-то нравится присутствие NULL в приложении.

На стадии проектирования и непосредственной разработки приложения я стараюсь любыми способами уменьшить количество содержащихся в нем значений NULL. Такое стремление отчасти объясняется тем, что проверка клиентским приложением существования значений NULL обычно занимает много времени и ресурсов. Например, при попытке объединения с помощью оператора JOIN двух таблиц, в столбцах которых присутствуют значения NULL, можно получить совершенно не тот результат, который ожидался. Выйти из такой ситуации можно, изменив тип оператора JOIN и повторно выполнив запрос, что, естественно, потребует дополнительных ресурсов на сервере.

Экскурс

Одним из способов, с помощью которых можно уменьшить количество значений NULL в приложении, является использование так называемых значений по умолчанию.

Значения по умолчанию, определяемые в качестве значений столбцов таблицы, обеспечивают автоматическую подстановку данных при внесении в таблицу новых строк информации. Например, при наличии в таблице поля OrderDate типа datetime, можно указать на необходимость автоматической подстановки в это поле текущей даты при внесении в таблицу новой строки информации.

При создании таблицы с помощью программы *Query Analyzer* для указания значения по умолчанию следует воспользоваться ключевым словом DEFAULT, в то время как при создании таблицы с помощью программы *Enterprise Manager* это значение задается с помощью параметра *Default Value* (Значение по умолчанию), расположенного в редакторе таблиц. Определение значения по умолчанию гарантирует ввод данных в поле таблицы даже в том случае, если по каким-либо причинам это поле было проигнорировано пользователем.

Другой способ, с помощью которого можно уменьшить количество значений NULL в приложении, — непосредственный запрет ввода в столбец таблицы значения NULL, что достигается установкой параметра NOT NULL. Попытка ввода в такой столбец значения NULL приведет к ошибке, и внесение информации в таблицу будет запрещено.

При разработке таблиц приложения SQLSpyNet мы непосредственно укажем для каждого из столбцов, разрешается или нет вводить в него значение NULL. Таким образом можно обеспечить целостность хранящейся в базе данных информации и исключить ситуацию появления неопределенных значений в самом неподходящем месте.

К сожалению, попытка максимально уменьшить количество значений NULL не всегда оканчивается безоговорочным успехом, причем в основном это вызвано вполне объективными причинами. Сказанное выше, например, справедливо для ситуации, в которой значения некоторых столбцов обновляются не сразу, а только по наступлении определенного момента, или же значение столбца попросту неизвестно во время внесения в таблицу данных.

Подводя итог, следует отметить: как бы вы ни старались избежать значений NULL в приложении, они неизбежно будут представлять собой неотъемлемую часть любой СУБД, и все, что вы можете сделать, — это просто попытаться максимальным образом уменьшить их число.

На заметку

При создании нового столбца SQL Server 2000 по умолчанию подразумевает для него установку параметра NOT NULL. Данное свойство этой СУБД отличает ее от стандарта ANSI SQL-92, в соответствии с которым, если только явно не указан параметр NOT NULL, для нового столбца по умолчанию устанавливается параметр NULL. При желании это свойство SQL Server 2000 можно изменить, согласовав его с указанным стандартом, однако будет лучше, если явным образом указать для каждого столбца параметр NULL или NOT NULL. Преимущество такого подхода в том, что при переходе на использование другой СУБД (даже и не думайте!) можно быть полностью уверенным в сохранении всех параметров столбцов.

- Щелкните на вкладке Columns (Столбцы), расположенной в нижней части диалогового окна редактора таблиц (см. рис. 3.17). Установите значение параметра Identity (Идентификационный столбец) равным Yes. В результате этого

параметрам Identity Seed (Начальное значение идентификационного столбца) и Identity Increment (Приращение значения идентификационного столбца) будет автоматически присвоено значение 1, что полностью совпадает с указанными выше требованиями.

- Для того чтобы определить первичный ключ таблицы, щелкните на столбце PersonID. В результате слева от названия столбца должна появиться стрелка, указывающая текущий столбец. После этого следует щелкнуть на пиктограмме с изображением ключа, расположенной на панели инструментов, или щелкнуть правой кнопкой мыши и выбрать команду Set Primary Key (Установить первичный ключ).

На заметку Хотя это вряд ли что-то скажет вам в данный момент, SQL Server 2000 автоматически создает уникальное индексирование всякого столбца, выступающего в роли первичного ключа. Более подробно индексирование рассматривается в главе 11, "Администрирование разведывательной сети".

- Введите оставшиеся имена столбцов и типы данных, как показано на рис. 3.17.
- И наконец, щелкните на пиктограмме сохранения таблицы (пиктограмма с изображением дискеты) и введите ее имя, в данном случае — **Person**. Заметьте: в качестве имени таблицы используется то же имя, что и при разработке диаграммы отношений между объектами. В отличие от такого принципа выбора имени, некоторые организации при именовании таблицы непременно указывают на то, что созданный объект базы данных представляет собой именно таблицу. Например, многие компании в данном случае назвали бы рассматриваемую таблицу *tblPerson*, *tPerson* или *PersonTable*. Если разобраться, то в этом нет большой необходимости. Используя SQL Server 2000, мы точно знаем, что *Person* — это таблица, поскольку она располагается в папке Tables. К тому же, как отмечалось ранее, имя таблицы должно быть по возможности кратким и понятным.
- Закройте диалоговое окно редактора таблиц. Созданная только что таблица должна появиться в папке Tables (если этого не произошло, щелкните на папке правой кнопкой мыши и выполните команду Refresh (Обновить)). Отметьте, что в расположенном справа от дерева объектов окне с описанием таблиц папки Tables таблица Person имеет тип User. Это означает, что данная таблица была создана непосредственно при участии пользователя (в нашем случае — sa), а не сгенерирована СУБД автоматически.

На заметку Таблицы типа *User* достаточно часто называют базовыми, поскольку они хранят пользовательские данные. Здесь может возникнуть небольшая путаница, так как некоторые разработчики привыкли называть базовыми таблицы самого высокого уровня в базе данных.

Экскурс

"...Роза пахнет розой, хоть розой назови ее, хоть нет..."

Многие компании, в которых я работал, и, естественно, множество других компаний по всему миру придерживаются одного раз и навсегда выбранного стандарта именования объектов базы данных, включая столбцы таблиц. В индустрии создания программного обеспечения это называется хорошим тоном и позволяет разработчикам без каких-либо усилий понять предназначение того или иного объекта базы данных.



Многие разработчики/компании в качестве имени первичного ключа таблицы `Person` выбрали бы *Person_ID* или *PersonID*. В каком бы месте базы данных (в нашем случае — SQLSpyNet) вам ни встретилось это имя, знайте, что оно ссылается на таблицу `Person`. Еще раз повторюсь, что подобный способ именования первичных ключей является очень хорошей практикой именования объектов базы данных. Тем не менее следует внимательно проследить, чтобы суффикс *ID* встречался только в именах первичных или внешних ключей таблиц.

Помните, что выбранное имя должно быть простым и осмысленным, избавьтесь в именах от совершенно ненужных определений типов, а также избегайте использования в качестве имен зарезервированных слов.

Приведенные выше правила именования применимы не только к столбцам таблиц, но и ко всем остальным объектам базы данных.

Итак, первая таблица создана. Анализируя предпринятые для этого действия, следует отметить, что создание таблицы — это чуть ли не самое простое занятие в процессе разработки приложения, использующего базу данных. Вдохновившись простотой создания таблиц, построим еще три таблицы с помощью графического интерфейса пользователя программы Enterprise Manager и четыре таблицы с помощью кода Transact-SQL, воспользовавшись при этом оператором `CREATE TABLE`.

Процесс создания таблицы `AddressType` практически ничем не отличается от процесса создания таблицы `Person`. На рис. 3.18 показано диалоговое окно редактора таблиц, в котором изображены все столбцы, типы данных, а также первичный ключ таблицы `AddressType`.

Рис. 3.18. Столбцы таблицы `AddressType`

Таблица AddressType представляет собой так называемую справочную таблицу. Это название, с одной стороны, оправдывается наличием в ней всего лишь нескольких столбцов, а с другой — ее положением в диаграмме отношений между объектами. Дело в том, что справочные таблицы располагаются как правило "на задворках" диаграммы отношений, между объектами; именно это и происходит с таблицей AddressType.

Таблица AddressType используется для хранения различных типов адресов (почтовый адрес, адрес проживания и т.д.), которые может иметь человек. Преимущество использования справочных таблиц заключается в том, что, если впоследствии потребуется добавить еще несколько типов адресов (например, служебный), это можно сделать без изменения модели данных.

Еще одним преимуществом использования справочных таблиц является то, что они помогают гарантировать целостность данных. При наличии справочной таблицы всегда можно быть уверенным в том, что, используя строку с идентификационным номером 5, мы получим информацию, касающуюся, например, домашнего адреса человека.

Справочные таблицы помогают предотвратить ввод информации, которая может быть неправильно истолкована из-за наличия в ней орфографических ошибок.

Экскурс

Целостность данных

Под целостностью данных подразумевается корректность и непротиворечивость хранящейся в базе данных информации. Если по каким-либо причинам в базе данных появляется несогласованная информация, то это в первую очередь указывает на отсутствие целостности данных.

SQL Server 2000 поддерживает, а значит, позволяет реализовать в приложении четыре основные категории целостности данных.

- Целостность на уровне таблиц (entity integrity) — самый низкий уровень целостности данных, который реализуется за счет использования первичных ключей и связанных индексов. Этот тип целостности запрещает ввод в базу данных информации о человеке с кодом налоговой инспекции *123456789*, если сведения о нем уже содержатся в базе данных.

- Целостность на уровне доменов (domain integrity) позволяет наложить ограничения на вносимые в столбец таблицы данные. Например, можно указать, что значение столбца *AnnualSalary* (Годовой доход) должно быть больше 10 000 долларов.

- Целостность на уровне ссылок (referential integrity) устанавливает правило, в соответствии с которым вносимое в столбец значение должно существовать в связанной таблице. Этот тип целостности реализуется посредством использования внешних ключей. Например, при вводе в базу данных информации о человеке и его адресе проживания (эта часть информации хранится в отдельной таблице) целостность на уровне ссылок требует, чтобы в таблице *Address* содержалась информация об адресе человека с соответствующей записью в таблице *Person*.

- Целостность, определяемая пользователем (user-defined integrity), позволяет разработчику установить собственные правила целостности, не вошедшие ни в одну из перечисленных категорий.

На основе упомянутых выше категорий целостности данных можно создавать новые правила целостности.

Другой способ, с помощью которого можно обеспечить непротиворечивость информации, — использование типов данных. Типы данных в SQL Server 2000

Экскурс

очень напоминают типы данных, поддерживаемые многими языками программирования. Так, в SQL Server 2000 есть строковый тип данных (такой, как `char` и `varchar` в языках программирования), тип для представления целого числа, денег, даты и многого другого. Кроме этого, в SQL Server 2000 появился новый тип данных, который называется `SQL_variant`. Те, кто знаком с языком программирования Visual Basic, наверняка увидят в этом типе данных много общего с типом данных `variant`, поддерживаемым Visual Basic. Тип данных `SQL_variant` позволяет хранить в столбце таблицы значения различных типов.

Используя типы данных, можно определить правило, в соответствии с которым в столбце таблицы должны храниться только значения, совместимые с заданным типом. Предположим, что столбец *DateOfBirth* был определен для хранения дат. Учитывая это ограничение, СУБД не разрешит пользователю ввести в такой столбец некорректное с позиции типа данных значение, например строку *unknown*.

Все перечисленные типы целостности данных будут реализованы при разработке приложения SQLSpyNet, что позволит разобраться на практике с вопросами обеспечения непротиворечивости информации.

Третья создаваемая таблица также является справочной. Практически все характеристики таблицы `Country` полностью совпадают с характеристиками таблицы `AddressType`, что позволяет добавлять новые страны без внесения изменений в модель данных. При создании таблицы `Country` примените тот же метод, который использовался при создании двух предыдущих таблиц. Обратитесь к рис. 3.19 за информацией о столбцах, типах данных и первичном ключе таблицы `Country`.

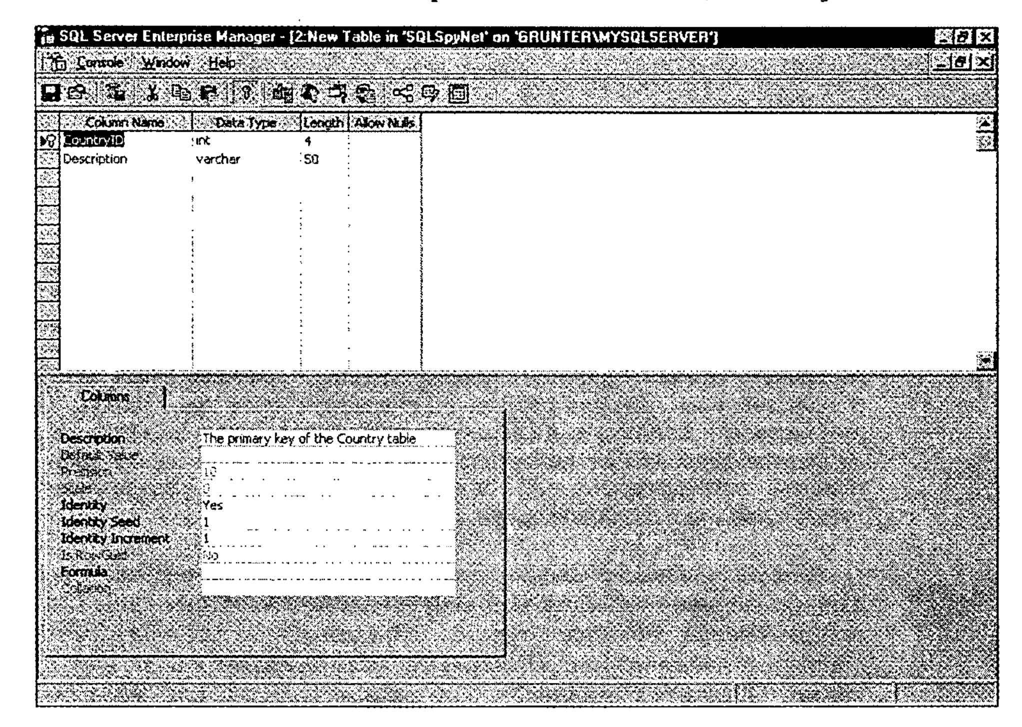


Рис. 3.19. Столбцы таблицы `Country`

Четвертой, и последней, таблицей, которая будет создана с помощью графического интерфейса пользователя, является Address. В ней хранятся различные адреса, которые может иметь человек.

Для того чтобы создать таблицу Address, воспользуйтесь тем же методом, который применялся при создании трех предыдущих таблиц. Обратитесь к рис. 3.20 за информацией о столбцах, типах данных и первичном ключе таблицы Address.

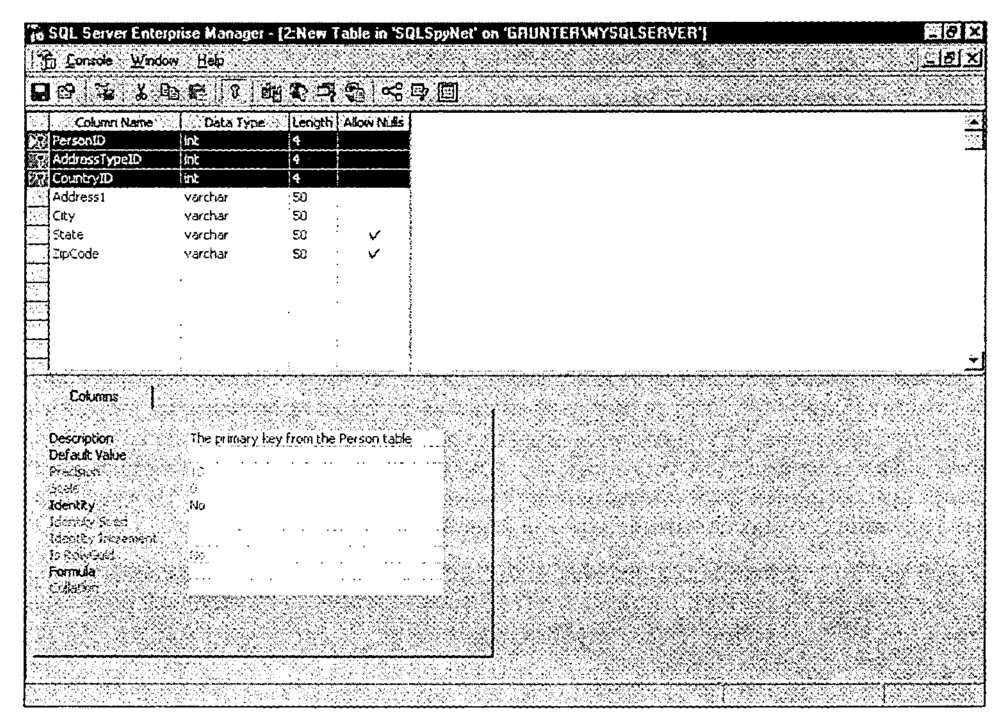


Рис. 3.20. Столбцы таблицы Address

Ключи, ключи и еще раз ключи!

Экскурс

Обратите внимание на способ, с помощью которого определяется первичный ключ таблицы Address. Такой ключ получил название *составного* первичного ключа. В данном случае составной первичный ключ ассоциативной таблицы Address состоит из внешних ключей трех таблиц. Преимуществом использования составного первичного ключа является дополнительное обеспечение целостности данных, которая в случае таблицы Address подразумевает, что одному человеку может соответствовать только один тип адреса для одной страны.

Термин

Составной первичный ключ (*composite primary key*) представляет собой первичный ключ таблицы, который состоит из более чем одного поля. Ассоциативная таблица Address, объединяющая таблицы Person, Country и AddressType, должна содержать внешние ключи всех трех таблиц. Обратите внимание, что ни один из внешних ключей — CountryID, AddressTypeID или PersonID — не может уникально характеризовать строку таблицы Address, поскольку один и тот же человек

Термин

(равно как и один и тот же адрес и одна и та же страна) может быть представлен в ней более одного раза. Единственно верным выходом из этой ситуации является объединение в качестве первичного ключа таблицы `Address` трех внешних ключей таблиц `Country`, `Person` и `AddressType`. Полученный таким образом составной первичный ключ будет уникально определять каждую строку таблицы `Address`.

Экскурс

Несмотря на то что описанный выше способ создания первичного ключа является в общем-то безупречным (к тому же этот способ настоятельно рекомендуется использовать с позиции реляционной теории), таблица `Address` могла бы иметь и другой тип первичного ключа. Все очень просто: необходимо всего лишь создать столбец `AddressID` и объявить его идентификационным столбцом таблицы `Address`. В контексте реляционной теории это будет вполне нормальный первичный ключ для ассоциативной таблицы. Множество разработчиков баз данных используют именно последний метод определения первичного ключа, но, хотя это может кому-то не понравиться, я утверждаю, что такой способ не обеспечивает целостности данных. На самом деле, такой способ определения первичного ключа не исключает возможности внесения в ассоциативную таблицу нескольких строк с одинаковым типом адреса, страной и человеком. Чтобы избежать подобной ситуации, можно воспользоваться уникальным индексированием соответствующих столбцов, однако в этом вряд ли есть какой-либо смысл, раз существует вполне корректный способ задания первичного ключа путем объединения внешних ключей таблиц.

Ради обеспечения целостности данных и ради соответствия классическим правилам реляционной теории мы создали таблицу `Address`, объявив в ней составной первичный ключ. И, несмотря на то что обновление данных в такой таблице потребует немножко больше изобретательности, я уверен, что в конечном итоге преимущества такого подхода перевесят его недостатки (которых на самом деле не так уж и много).



Удерживая нажатой клавишу <Ctrl>, можно выделить одновременно более одного столбца таблицы.

Входящие в состав первичного ключа таблицы `Address` первичные ключи таблиц `Person`, `Country` и `AddressType` называются *внешними ключами* таблицы `Address`. Использование внешних ключей — первый шаг к созданию отношений между таблицами. Второй шаг заключается в их непосредственном определении.

Экскурс

Где же ты, 3НФ?

Посмотрев чуть более внимательно на структуру таблицы `Address`, можно заметить, что она на самом деле не удовлетворяет требованиям третьей нормальной формы (3НФ). Главной причиной является наличие в таблице так называемой *транзитивной зависимости*. Попробуйте обнаружить транзитивно зависимые столбцы таблицы `Address` самостоятельно (ответ на этот вопрос будет дан в конце главы).

Термин

Транзитивная зависимость (*transitive dependency*) — это наличие некоторой связи между не являющимися ключевыми атрибутами таблицы. Примером транзитивной зависимости в таблице может быть связь между почтовым индексом и названием города.

Определение связей между таблицами путем создания диаграммы базы данных

Выделив папку Diagrams (Диаграммы), щелкните на ней правой кнопкой мыши и выполните команду New Database Diagram (Новая диаграмма базы данных), в результате чего должно открыться диалоговое окно мастера Database Diagram Wizard (Мастер создания диаграмм базы данных) (если этого не произошло, не волнуйтесь, поскольку мы все равно собираемся создать диаграмму базы данных без использования соответствующего мастера). Щелкните на кнопке Cancel (Отменить). Вы увидите окно редактора диаграмм, содержащее пустую область. Щелкните на ней правой кнопкой мыши и выполните команду Add Table (Добавить таблицу), как показано на рис. 3.21.

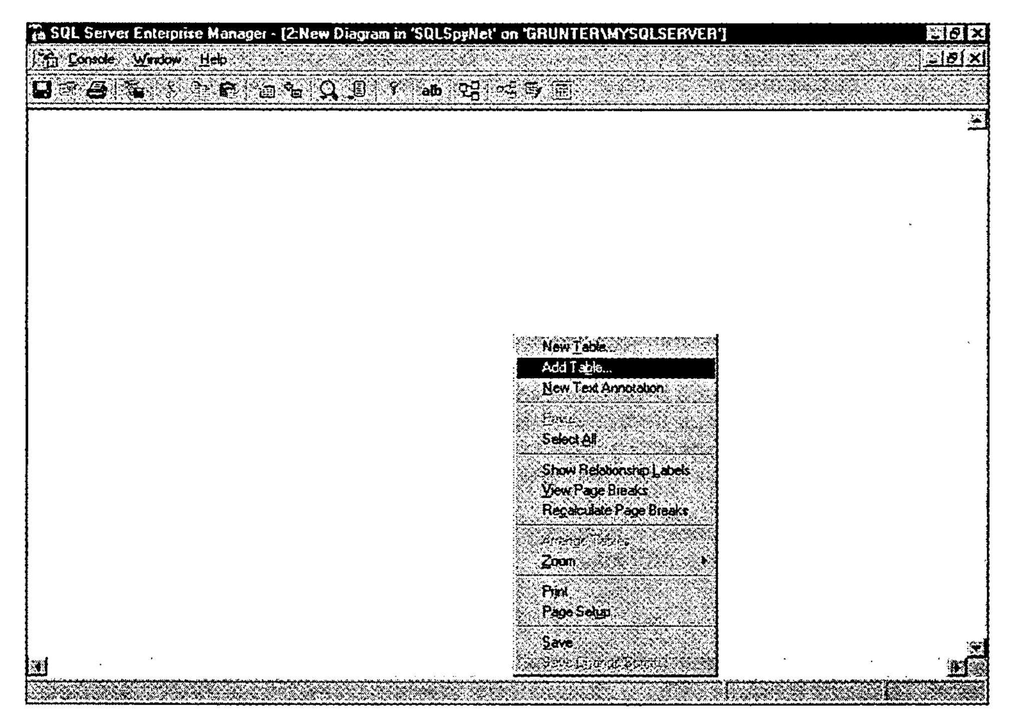


Рис. 3.21. Добавление таблицы к диаграмме базы данных

На заметку Редактор диаграмм представляет собой чрезвычайно мощное средство управления базой данных. С его помощью можно выполнять множество различных операций, включая создание диаграмм, определение отношений между таблицами, изменение столбцов и даже создание новых таблиц. Кроме этого, следует отметить простоту использования редактора диаграмм, что вполне заслуженно делает его чуть ли не основным средством управления базой данных для многих разработчиков.

В результате выполнения команды Add Table появится одноименное диалоговое окно (рис. 3.22), в котором содержится список всех таблиц базы данных, включая системные.

Добавьте к диаграмме базы данных только что созданные таблицы (`Address`, `AddressType`, `Country` и `Person`). Для этого выберите их имена из списка и щелкните на кнопке Add (Добавить).

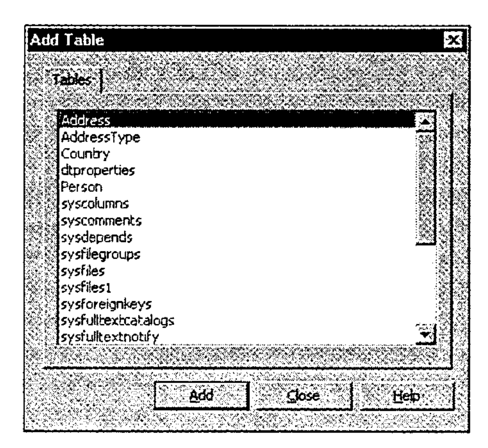


Рис. 3.22. Выберите таблицы, которые необходимо добавить к диаграмме базы данных

Добавьте к диаграмме базы данных только что созданные таблицы (`Address`, `AddressType`, `Country` и `Person`). Для этого выберите их имена из списка и щелкните на кнопке Add (Добавить).



Удерживая нажатой клавишу <Ctrl>, можно выделить одновременно более одной таблицы.

Добавив к диаграмме базы данных все четыре таблицы, закройте диалоговое окно Add Tables.

Переместите таблицы в области диаграммы базы данных таким образом, чтобы они все размещались в пределах одного окна.

Создание отношений между таблицами

Создание отношений между таблицами базы данных SQLSpyNet начнем с определения отношений между таблицами `Person` и `Address`. Щелкните на пиктограмме с изображением ключа, расположенной возле столбца `PersonID` таблицы `Person`, и перетащите ее к аналогичной пиктограмме, расположенной рядом с одноименным столбцом таблицы `Address`. Появится диалоговое окно Create Relationship (Создание отношения), показанное на рис. 3.23.

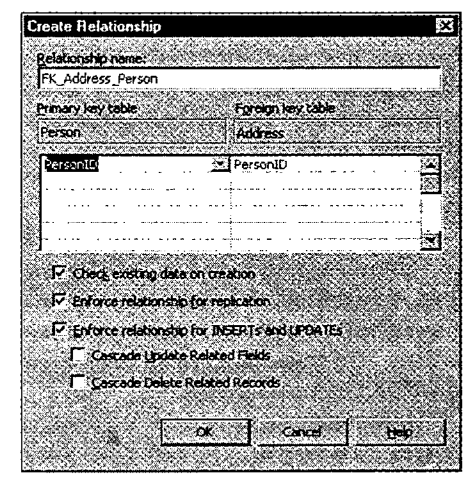


Рис. 3.23. Диалоговое окно Create Relationship

С помощью элементов управления диалогового окна Create Relationship можно осуществить довольно мощные операции над базой данных; помимо этого, в окне представлено одно интересное нововведение SQL Server 2000.

Начнем изучать элементы управления диалогового окна Create Relationship с его верхней части. Вспомните, что каждый объект базы данных в SQL Server 2000, включая отношение между таблицами, должен быть однозначно определяем. Обратите внимание на то, что SQL Server 2000 автоматически подбирает имя для создаваемого отношения между таблицами. Следует отметить, что, несмотря на невозможность изменения имени отношения, используемое SQL Server 2000 правило именования вполне удовлетворяет нашим основным требованиям.

Действительно, используемое по умолчанию имя (`FK_Address_Person`) описывает отношение между таблицами достаточно наглядно: `FK` — это аббревиатура от *foreign key* (внешний ключ), а `Address` и `Person` — соответственно имена участвующих в отношении таблиц `Address` (отмечена пиктограммой, изображающей символ бесконечности) и `Person` (отмечена пиктограммой с изображением ключа).

Следующим шагом является определение столбцов таблиц, непосредственно участвующих в отношении.

Первичным ключом данного отношения является столбец `PersonID` таблицы `Person` (первичный ключ этой таблицы). Внешним ключом отношения выступает столбец `PersonID` таблицы `Address` (внешний ключ этой таблицы). При наличии большего количества столбцов, определяющих отношение между таблицами, их необходимо внести в соответствующие части таблицы диалогового окна Create Relationship. В данном случае отношение между таблицами полностью задается двумя столбцами `PersonID`, как показано на рис. 3.23, а поэтому определение участвующих в отношении столбцов таблиц можно считать завершенным.

Следующим элементом управления диалогового окна Create Relationship является флажок Check existing data on creation (Проверять существование данных при создании отношения). Установка этого флажка позволяет создавать отношение между таблицами на основе существования данных в таблице с внешним ключом. Поскольку мы имеем дело с только что созданными таблицами, не заполненными пока никакой информацией, сброс/установка данного флажка никоим образом не повлияет на создание отношения между таблицами. Тем не менее оставьте флажок Check existing data on creation в его начальном положении, т.е. установленным.

Установка флажка Enforce relationship for replication (Сохранять отношение при репликации) гарантирует сохранение целостности данных на уровне ссылок (ограничение внешнего ключа) при репликации таблицы в другую базу данных. Несмотря на то что проблемы репликации, мягко говоря, не очень-то волнуют на текущей стадии разработки приложения, рано или поздно мы придем к такой ситуации, когда будет крайне необходимо обеспечить именно подобный тип целостности данных, а потому убедитесь, что флажок Enforce relationship for replication установлен.

Флажок Enforce relationship for INSERTs and UPDATEs (Сохранять отношение между таблицами при вставке и обновлении информации) представляет особую ценность. Установка этого флажка гарантирует наличие записи об определенном человеке в таблице `Person` при вставке или обновлении информации о нем в таблице `Address`. Обязательно убедитесь в том, что флажок Enforce relationship for INSERTs and UPDATEs установлен.

И наконец, мы подошли к упомянутому выше нововведению SQL Server 2000. Следует отметить, что эта функциональная особенность была доступна разработчикам Microsoft Access в течение долгого времени, так что разработчикам SQL Server скрепя сердце приходилось признавать, что Microsoft Access имеет по

крайней мере одно преимущество перед SQL Server. Спешу вас обрадовать: это "смутное" время закончилось!

- Установка флажка **Cascade Update Related Fields** (Каскадное обновление связанных полей) указывает SQL Server 2000 на необходимость обновлять значение внешнего ключа таблицы `Address` всякий раз, когда изменяется соответствующее ему значение первичного ключа таблицы `Person`. Например, если в таблице `Address` хранилась запись о человеке с идентификационным номером 5 и по какой-то причине значение первичного ключа (идентификационный номер) в таблице `Person` изменилось с 5 на 7, то SQL Server 2000 автоматически обновит соответствующую этому первичному ключу запись в таблице `Address`. Довольно мощно, не правда ли?
- Установка флажка **Cascade Delete Related Fields** (Каскадное удаление связанных полей) указывает SQL Server 2000 на необходимость удаления всех записей в таблице с внешним ключом при удалении соответствующей записи из таблицы с первичным ключом. Например, если в таблице `Address` (таблица с внешним ключом) хранились записи об адресах определенного человека и по каким-то причинам информация о нем была удалена из таблицы `Person` (таблица с первичным ключом), то SQL Server 2000 автоматически удалит и записи обо всех адресах этого человека из таблицы `Address`. Следуя правилам классической реляционной теории (до того, как эта функциональная особенность появилась в SQL Server), прежде чем удалить запись с первичным ключом (родительскую запись), необходимо было удалить все записи с соответствующим внешним ключом (дочерние записи). При невыполнении указанного требования SQL Server выдавал сообщение об ошибке и запрещал дальнейшее проведение процедуры удаления записей.

Поскольку мы не хотим, чтобы нечаянное удаление записи о человеке из таблицы `Person` привело к удалению из базы данных всей касающейся этого человека информации, не устанавливайте флажок **Cascade Delete Related Fields**. Аналогично следует поступить и с флажком **Cascade Update Related Fields**, так как значение первичного ключа таблицы `Person` невозможно изменить вручную (по крайней мере, это не так просто), поскольку данный столбец является идентификационным столбцом таблицы.

После установки все необходимых параметров щелкните на кнопке OK, в результате чего SQL Server 2000 отобразит на диаграмме базы данных отношение между двумя таблицами. Обратите внимание на то, что таблица с внешним ключом отмечается символом бесконечности (∞), а таблица с первичным ключом — символом ключа. Между таблицами `Person` и `Address` устанавливается отношение типа "один ко многим" (рис. 3.24).

На заметку Символ * рядом с названием таблиц обозначает, что соответствующие изменения пока еще не были внесены в базу данных. Для того чтобы это сделать, необходимо подтвердить внесение изменений, сохранив диаграмму на диске.

Аналогичные действия следует повторить для двух оставшихся таблиц. Перетащите первичный ключ таблицы `AddressType` к внешнему ключу с таким же названием в таблице `Address`. В открывшемся диалоговом окне выберите те же самые параметры, что и для первого отношения (за исключением, естественно, имен столбцов).

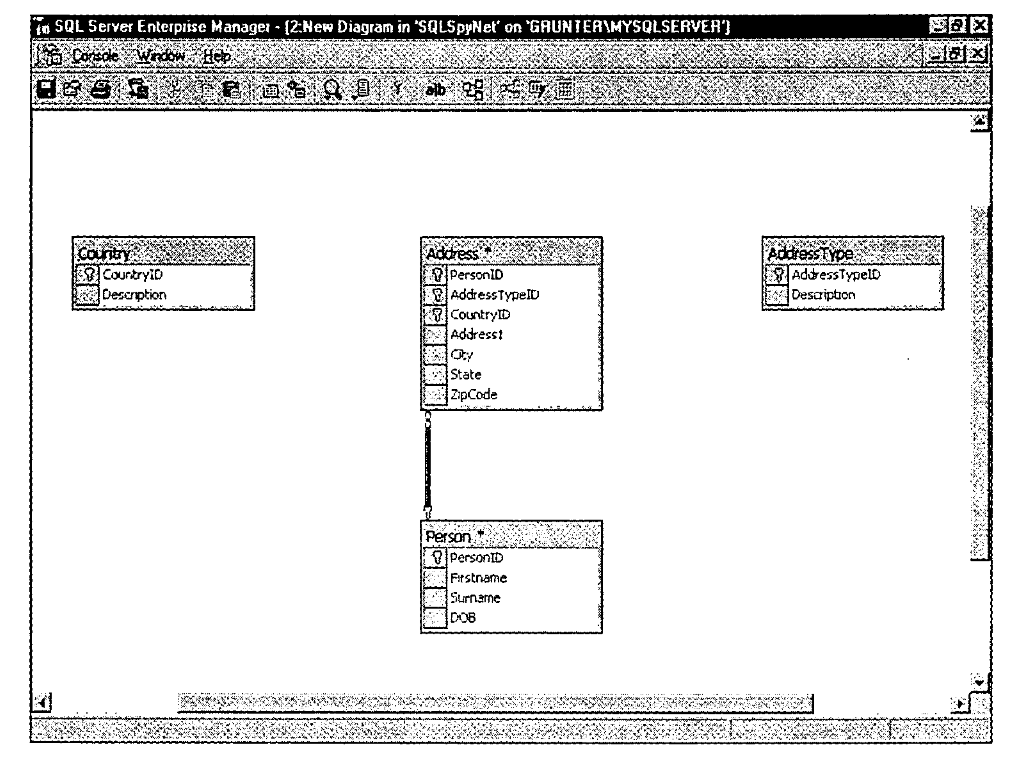


Рис. 3.24. Взаимоотношение между таблицами `Person` *и* `Address`

Повторите указанные действия по отношению к таблицам `Country` и `Address`. Получившаяся в результате диаграмма базы данных должна выглядеть так, как на рис. 3.25.

Для того чтобы сохранить изменения, внесенные в диаграмму базы данных, следует щелкнуть на пиктограмме с изображением дискеты. При первом сохранении диаграммы SQL Server 2000 попросит указать ее имя. Естественно, можно выбрать любое сколько-нибудь значимое имя, однако я рекомендую подойти к выбору имени диаграммы более серьезно, назвав ее, например, *Current ERD* (Текущая диаграмма отношений между объектами базы данных).

Щелкнув на кнопке OK, вы увидите новое диалоговое окно, в котором SQL Server 2000 предупредит о сохранении в базе данных некоторых таблиц (рис. 3.26).

При желании список подвергнутых изменениям таблиц можно сохранить в текстовом файле. Как правило, это довольно неплохая идея в тех случаях, когда приложение находится в устойчивом состоянии (т.е. не подвержено частым корректировкам) и когда необходимо вести учет всех внесенных в базу данных изменений. Поскольку приложение SQLSpyNet не нуждается в отслеживании внесенных в него изменений и пока что не является устойчивым, щелкните на кнопке Yes (Да).

После щелчка на кнопке Yes SQL Server 2000 выполнит все необходимые преобразования базы данных, соответствующие внесенным в нее изменениям. Вот и все! Вы только что создали самую что ни на есть настоящую реляционную базу данных. Примите мои поздравления по этому, несомненно, радостному для вас поводу!

Рис. 3.25. Диаграмма базы данных наглядно иллюстрирует все существующие отношения между ее таблицами

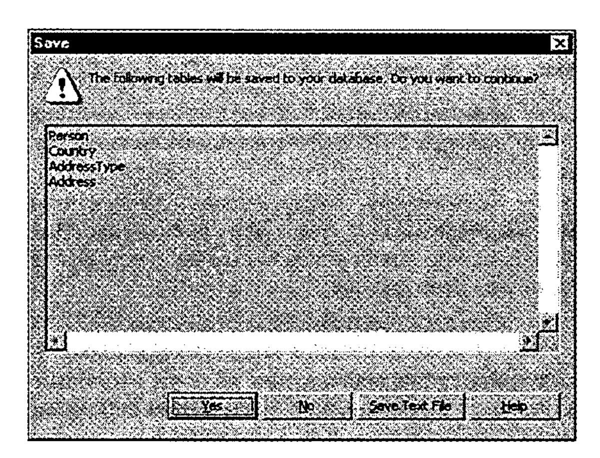


Рис. 3.26. С помощью этого диалогового окна SQL Server 2000 предупреждает пользователя о сохранении в базе данных некоторых таблиц

Создание таблиц с помощью кода Transact-SQL

Спроектировав добрую половину всех таблиц базы данных SQLSpyNet с помощью графического интерфейса пользователя, создадим оставшуюся часть таблиц с помощью кода Transact-SQL.

В данном разделе рассматриваются такие распространенные операторы Transact-SQL, как CREATE TABLE и ALTER TABLE.

Синтаксис оператора CREATE TABLE достаточно прост для понимания (по крайней мере, это касается создаваемых таблиц!). Придерживаясь определенного выше правила, начнем с родительских таблиц, а уж затем перейдем к дочерним таблицам, это будет гарантировать существование всех нужных таблиц на этапе определения отношений между ними.

Оператор языка Transact-SQL CREATE TABLE имеет много параметров, включая группу файлов, к которой будет отнесена таблица, владельца таблицы, первичный и внешний ключи таблицы, а также типы данных и имена столбцов.

Следует отметить, что в этой книге рассматривается только небольшая часть параметров оператора CREATE TABLE. В первую очередь это связано с тем, что многие параметры оператора CREATE TABLE при повседневном администрировании базы данных обычно не используются, что объясняется узкой областью их применения (например, они могут применяться в связи со сложными вопросами хранения данных, а также при определении репликации).

Единственной таблицей самого верхнего уровня среди четырех оставшихся таблиц базы данных SQLSpyNet является родительская таблица ActivityType. Она представляет собой справочную таблицу, хранящую информацию о типах деятельности, в которых могут участвовать тайные агенты и злоумышленники. Подобно всякой справочной таблице, она позволяет добавлять новые типы деятельности без внесения изменений в модель данных.

На заметку

Оставшиеся таблицы базы данных SQLSpyNet будут созданы с помощью оператора Transact-SQL CREATE TABLE. Аналогично тому как это делалось с помощью графического интерфейса пользователя, при создании данных таблиц определим только ограничения первичного ключа, оставив задачу определения ограничений внешнего ключа “на потом”. Для того чтобы задать ограничения внешнего ключа, определив тем самым отношения между таблицами, воспользуемся оператором Transact-SQL ALTER TABLE.

Итак, приступим к созданию таблицы ActivityType. Запустите программу Query Analyzer и выберите базу данных SQLSpyNet из расположенного в строке меню раскрывающегося списка.



Убедитесь лишний раз в том, что выбранная база данных — именно SQLSpyNet. В случае ошибки SQL Server 2000 создаст таблицу ActivityType в какой-то другой базе данных.

Таблица ActivityType состоит всего лишь из двух столбцов — ActivityTypeID и Description. Введите код, приведенный в листинге 3.3, в окно ввода кода программы Query Analyzer.

Листинг 3.3. Создание таблицы ActivityType с помощью кода Transact-SQL

Код для запуска

```
CREATE TABLE ActivityType
        (
        ActivityTypeID INT IDENTITY(1,1)NOT NULL
            CONSTRAINT PK_ActivityTypeID
            PRIMARY KEY CLUSTERED,
        Description VARCHAR(50)NOT NULL
        )
```

Выполнение этого кода приведет к созданию таблицы ActivityType в базе данных SQLSpyNet. В качестве результата Query Analyzer возвратит сообщение, подобное изображенному на рис. 3.27.

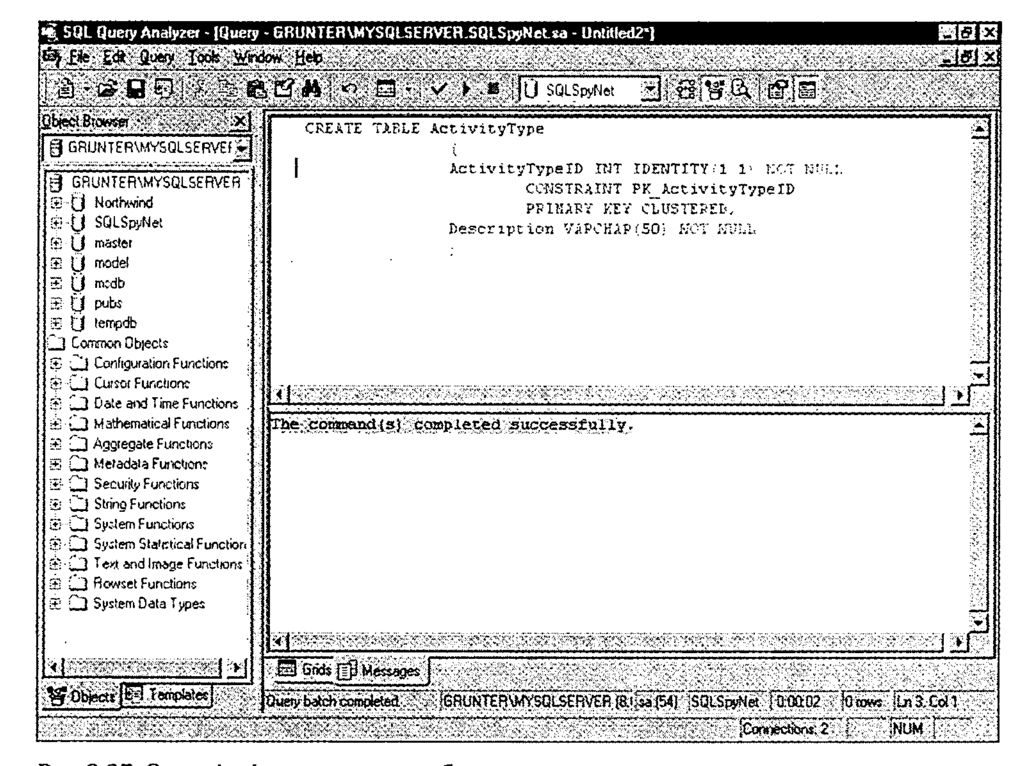


Рис. 3.27. Query Analyzer рапортует об успешном выполнении оператора CREATE TABLE

На заметку Если после выполнения приведенного в листинге 3.3 кода вы получили сообщение, отличное от приведенного выше, проверьте синтаксис оператора CREATE TABLE и попытайтесь выполнить его снова.

Возвратитесь к программе Enterprise Manager (или воспользуйтесь окном Object Browser программы Query Analyzer) и обновите содержимое папки Tables базы данных SQLSpyNet. В списке таблиц должна появиться только что созданная таблица ActivityType.

На заметку Несмотря на то что таблица ActivityType присутствует в списке таблиц базы данных SQLSpyNet, ее пока нет в созданной ранее диаграмме *Current ERD*. Для того чтобы она там появилась, следует выполнить действия, аналогичные тем, которые выполнялись при внесении в эту диаграмму предыдущих таблиц.

Проанализируем код, приведенный в листинге 3.3.

Анализ

- В строке 1 приведен оператор CREATE TABLE, указывающий Query Analyzer на необходимость создания новой таблицы с именем ActivityType.

Анализ

- Расположенная в строке 2 открывающая круглая скобка указывает на начало определения параметров оператора CREATE TABLE.
- В строке 3 кода Transact-SQL определяется первый столбец таблицы ActivityType. Это столбец типа INT, носящий имя ActivityTypeID. Столбец ActivityTypeID является идентификационным столбцом таблицы ActivityType. Расположенный в конце этой строки оператор NOT NULL определяет ограничение, в соответствии с которым столбец ActivityTypeID не может хранить значение NULL. Вспомните, что подобное ограничение накладывается на все идентификационные столбцы и первичные ключи таблиц.
- В строке 4 определяется имя ограничения первичного ключа таблицы ActivityType. Данная строка может быть опущена, в результате чего SQL Server 2000 автоматически сгенерирует стандартное имя ограничения первичного ключа (аналогично тому, как SQL Server 2000 генерировал имя ограничения внешнего ключа). Несмотря на такую возможность, мы определяем собственное имя ограничения первичного ключа — PK_ActivityTypeID (PK является аббревиатурой от *primary key* — первичный ключ).
- Строка 5 указывает на то, что первичный ключ таблицы ActivityType был определен в строке 2. Для того чтобы понять, как это могло произойти, следует обратить внимание на то, каким образом заканчивается строка 5. Все дело в том, что строка 5 оканчивается символом запятой (,), который, как уже отмечалось ранее, обозначает конец определения столбца таблицы. Ну а поскольку запятая стоит после оператора PRIMARY KEY CLUSTERED и, в свою очередь, не отделена другой запятой от строки 3, то столбец ActivityTypeID автоматически становится первичным ключом таблицы ActivityType. Ключевое слово CLUSTERED устанавливает порядок физического хранения данных на диске, который в этом случае полностью определяется значением первичного ключа (1, 2, 3, 4 и т.д.). Аналогом такого способа хранения информации выступает десятичная система Дьюи (Dewey), которая встречается в библиотеках, где книги упорядочиваются и хранятся в соответствии с их идентификационным номером. Если ключевое слово CLUSTERED будет опущено, то SQL Server 2000 создаст для первичного ключа так называемый кластеризованный индекс, обеспечив при этом его уникальность в пределах данной таблицы. Более подробно кластеризованные индексы рассматриваются в главе 11, "Администрирование разведывательной сети".
- В строке 6 определяется второй столбец таблицы ActivityType, который носит имя Description. Он предназначен для хранения данных типа VARCHAR длиной до 50 символов и, подобно столбцу ActivityTypeID, не может хранить значение NULL.
- Расположенная в строке 7 закрывающая круглая скобка обозначает конец определения параметров оператора CREATE TABLE и соответствует открывающей круглой скобке в строке 2.

Рассмотрев один из простейших примеров кода Transact-SQL, создающего таблицу, перейдем непосредственно к созданию оставшихся трех таблиц базы данных SQLSpyNet. Для того чтобы создать таблицу Spy, введите код, приведенный в листинге 3.4.

Листинг 3.4. Код Transact-SQL, создающий таблицу Spy

Код для запуска

```
1: CREATE TABLE Spy
2:         (
3:         SpyID INT IDENTITY(1,1)NOT NULL
4:             CONSTRAINT PK_SpyID
5:             PRIMARY KEY CLUSTERED,
6:         PersonID INT NOT NULL,
7:         SpyNumber VARCHAR(10)NULL,
8:         Alias VARCHAR(25)NULL,
9:         DateCommencedWork DATETIME NOT NULL
10:             DEFAULT (GETDATE()),
11:         AnnualSalary MONEY NOT NULL
12:             CONSTRAINT Check_AnnualSalary
13:             CHECK (AnnualSalary >=10000),
14:         IsActive BIT NOT NULL
15:             DEFAULT 1
16:         )
```

Анализ

Объявление таблицы Spy во многом подобно объявлению предыдущей таблицы, за исключением наличия некоторых новых параметров.

Оператор `DEFAULT` указывает SQL Server 2000 на необходимость автоматической подстановки в столбец определенного значения в том случае, если по каким-либо причинам значение этого столбца не было указано явно. Так, расположенный в строке 10 оператор задает значение по умолчанию для столбца `DateCommencedWork`, которое в данном случае представляет собой текущую дату.

Функция `GETDATE()` — специальная, возвращающая в качестве результата текущую дату и время сервера, на котором уставлена СУБД SQL Server 2000. Более подробно функция `GETDATE()` рассматривается в главе 6, "Использование функций для повышения эффективности управления информацией".

Значение по умолчанию определяется также и для столбца `IsActive`. В том случае, если значение столбца `IsActive` не будет указано явно, SQL Server 2000 автоматически установит его равным 1.

На заметку

Тип данных `BIT` используется, как правило, для представления булевых значений (правда/ложь). Поля данного типа способны хранить только два значения: 0 (что соответствует лжи) либо 1 (что соответствует правде). Стоит отметить, что имя столбца `IsActive` — характерный пример негласно принятого правила именования столбцов, хранящих булев тип данных.

В строке 13 определяется еще один тип целостности данных. С помощью оператора `CHECK` устанавливается нижняя граница для вводимых в столбец `AnnualSalary` значений, которая равна 10 000 долларов (если задуматься, сущие копейки для международного шпиона!).

Возможность определения ограничений на вводимые в столбец таблицы данные — одно из главных преимуществ и наглядный пример гибкости настройки SQL Server 2000. Благодаря этой возможности довольно просто определить любое требуемое ограничение (например, ограничение по заработной плате), а при необходимости с не меньшей легкостью изменить его.

Разобравшись с таблицей `Spy`, перейдем к "противоположной" (в некотором смысле) таблице — `BadGuy`. Итак, введите и выполните код, приведенный в листинге 3.5.

Листинг 3.5. Создание таблицы BadGuy с помощью кода Transact-SQL

Код для запуска

```
CREATE TABLE BadGuy
        (
        BadGuyID INT IDENTITY(1,1)NOT NULL
            CONSTRAINT PK_BadGuyID
            PRIMARY KEY CLUSTERED,
        PersonID INT NOT NULL,
        KnownAs VARCHAR(25)NULL,
        IsActive BIT NOT NULL
            DEFAULT 1
        )
```

Поскольку в отношении использования различных особенностей SQL Server 2000 таблица BadGuy ничем не отличается от таблицы Spy (на самом деле она даже немного проще), не будем анализировать приведенный выше код, а перейдем непосредственно к созданию четвертой, и последней, таблицы базы данных SQLSpyNet — Activity.

Введите и выполните код, приведенный в листинге 3.6.

Листинг 3.6. Код Transact-SQL, создающий таблицу Activity

Код для запуска

```
CREATE TABLE Activity
        (
        ActivityID INT IDENTITY(1,1)NOT NULL
            CONSTRAINT PK_ActivityID
            PRIMARY KEY CLUSTERED,
        SpyID INT NOT NULL,
        BadGuyID INT NOT NULL,
        ActivityTypeID INT NOT NULL,
        IsPlanFoiled BIT NOT NULL
            DEFAULT 0,
        DatePlanAttempted DATETIME NOT NULL
            DEFAULT (GETDATE()),
        DatePlanFoiled DATETIME NULL
        )
```

На заметку

Наверняка вы заметили, что определение ассоциативной таблицы Activity несколько отличается от определения созданной ранее ассоциативной таблицы ActivityType. Оно связано с наличием собственного первичного ключа — столбца ActivityID. Напомним, что ранее мы отказались от такого способа определения первичного ключа в пользу более эффективного обеспечения целостности данных. Зачем же реализовать этот неэффективный метод в таблице Activity? Основной причиной является желание наглядно проиллюстрировать оба способа создания первичного ключа и предоставить вам возможность самим решить, какой из этих способов, на ваш взгляд, лучше. К сожалению, реализуя второй способ определения первичного ключа, для обеспечения целостности данных следует внести в таблицу Activity несколько дополнительных корректив, которые заключаются в создании так называемого уникального индексирования (во многом аналогично созданию первичного ключа).

Как было сказано выше, для того чтобы обеспечить целостность данных, следует выполнить несколько дополнительных операций. Введите и выполните код Transact-SQL, приведенный в листинге 3.7.

Листинг 3.7. Код Transact-SQL, создающий уникальное индексирование таблицы Activity

Код для запуска

```
1: CREATE UNIQUE NONCLUSTERED INDEX IDX_Spy_BadGuy_ActType
2:    ON Activity (SpyID, BadGuyID, ActivityTypeID)
```

Проведем анализ кода, приведенного в листинге 3.7.

Анализ

- В строке 1 указывается необходимость создания уникального индексирования с именем IDX_Spy_BadGuy_ActType (индексирование по столбцам SpyID, BadGuyID и ActivityTypeID). Данное индексирование не должно быть кластеризованным, так как ранее при создании таблицы определен кластеризованный первичный ключ. Поскольку кластеризованный индекс определяет способ физического размещения данных на диске, SQL Server 2000 разрешает определять только один такой индекс на таблицу. Действительно, если бы иметь два кластеризованных индекса, то какой из способов размещения данных на диске следовало бы принять за основной?
- В строке 2 указывается таблица, в которой необходимо создать уникальное индексирование. В данном случае это таблица Activity. Список имен столбцов, заключенный в круглые скобки, определяет уникальные столбцы таблицы Activity. Почему же в этом списке нет первичного ключа? Если первичный ключ будет присутствовать в списке однозначно определяемых столбцов, то необходимость в уникальном индексировании просто исчезнет, так как тогда каждое значение из этого списка будет "единственным и неповторимым" за счет присутствия в нем первичного ключа, который, напомним, является идентификационным столбцом таблицы.

Уникальный индекс следует также определить для столбца PersonID в таблицах Spy и BadGuy, поскольку единственный способ создания отношения типа "один к одному" заключается в уникальном индексировании столбца, выполняющего роль внешнего ключа в таблице с внешним ключом. Зачем это нужно? Ответ прост: один тайный агент может быть только одним человеком, а один человек, в свою очередь, не может быть более чем одним тайным агентом — вот отсюда-то и возникает отношение типа "один к одному". Аналогичные размышления полностью применимы и к таблице BadGuy.

Введя и выполнив код, приведенный в листинге 3.8, практически полностью завершим формирование таблицы Spy.

Листинг 3.8. Код Transact-SQL, создающий уникальный индекс для таблицы Spy

Код для запуска

```
CREATE UNIQUE NONCLUSTERED INDEX IDX_Spy_Person
   ON Spy (PersonID)
```

А теперь введите и выполните аналогичный код для таблицы BadGuy (листинг 3.9).

Листинг 3.9. Код Transact-SQL, создающий уникальный индекс для таблицы BadGuy

Код для запуска

```
CREATE UNIQUE NONCLUSTERED INDEX IDX_BadGuy_Person
    ON BadGuy (PersonID)
```

Выполнив приведенный выше код, вы закончите создание всех таблиц базы данных SQLSpyNet, а также определите *некоторые* отношения и индексы. Всего база данных SQLSpyNet состоит из восьми пользовательских (базовых) таблиц. Для того чтобы просмотреть список таблиц, воспользуйтесь деревом объектов программы Enterprise Manager или окном Object Browser программы Query Analyzer.

Прежде чем завершить работу по созданию таблиц базы данных SQLSpyNet, следует определить все существующие между ними отношения (аналогично тому, как это было сделано для таблиц Person и Address).

Итак, определим все оставшиеся отношения между таблицами и покончим с этим наверняка уже наскучившим вам занятием! Уверен, что, посмотрев на диаграмму отношения между объектами базы данных SQLSpyNet, вы без труда создадите все недостающие отношения с помощью редактора диаграмм. Однако это будет слишком просто и вряд ли добавит новых знаний и опыта. Вместо этого для создания отношений между таблицами базы данных SQLSpyNet воспользуемся оператором Transact-SQL ALTER TABLE.

Первым отношением, определенным с помощью оператора ALTER TABLE, будет отношение между таблицами Spy и Person. Для определения отношения между этими таблицами следует ввести и выполнить код Transact-SQL в программе Query Analyzer, приведенный в листинге 3.10.

Листинг 3.10. Внесение изменений в таблицу Spy с целью создания отношения с таблицей Person

Код для запуска

```
1: ALTER TABLE Spy
2:     ADD CONSTRAINT FK_Spy_Person
3:         FOREIGN KEY (PersonID)
4:         REFERENCES Person(PersonID)
```

Анализ

Проанализируем приведенный код.

- Расположенный в строке 1 оператор ALTER TABLE указывает SQL Server 2000 на необходимость изменения определения таблицы Spy.
- В строке 2 явным образом указывается необходимость добавления в таблицу Spy нового ограничения с именем FK_Spy_Person (внешний ключ, связывающий таблицы Spy и Person).
- В строке 3 определяется тип ограничения FK_Spy_Person. В данном случае это ограничение, обеспечивающее целостность на уровне ссылок базы данных SQLSpyNet (которая, напомним, достигается путем определения внешнего ключа). Стол-

Анализ

бец PersonID таблицы Spy является внешним ключом, определяющим отношение между таблицами Spy и Person.

- В строке 4 указывается, что созданный внешний ключ таблицы Spy будет ссылаться на столбец PersonID таблицы Person.

Вот и все. Только что определено отношение между таблицами Person и Spy. Обратите внимание, что именно такие действия провел бы редактор диаграмм в ответ на соединение с помощью мыши столбцов PersonID таблиц Spy и Person. Теперь понимаете, насколько мощное это средство управления базой данных?

Создание отношения между таблицами Person и Spy рассматривается несколько позже, а пока что определим все оставшиеся связи между таблицами базы данных SQLSpyNet.

Следующим определяемым отношением будет отношение между таблицами BadGuy и Person. Необходимо отметить, что оно практически аналогично отношению между таблицами Spy и Person (по большому счету, можно просто скопировать приведенный в листинге 3.10 код и изменить в нем название таблицы Spy на BadGuy). Введите и выполните код, приведенный в листинге 3.11.

Листинг 3.11. Внесение изменений в таблицу BadGuy с целью создания отношения с таблицей Person

Код для запуска

```
ALTER TABLE BadGuy
    ADD CONSTRAINT FK_BadGuy_Person
        FOREIGN KEY (PersonID)
        REFERENCES Person(PersonID)
```

Следующим шагом будет определение ограничения внешнего ключа таблицы Activity.

Напомним, что таблица Activity связана с тремя остальными таблицами (Spy, BadGuy и ActivityType), так что потребуется определить соответственно три ограничения внешнего ключа.

Введите и выполните код, приведенный в листинге 3.12.

Листинг 3.12. Внесение изменений в таблицу Activity с целью создания отношений с таблицами Spy, BadGuy и ActivityType

Код для запуска

```
ALTER TABLE Activity
    ADD CONSTRAINT FK_ActivitySpy
        FOREIGN KEY (SpyID)
        REFERENCES Spy(SpyID),
        CONSTRAINT FK_Activity_BadGuy
            FOREIGN KEY (BadGuyID)
            REFERENCES BadGuy(BadGuyID),
        CONSTRAINT FK_Activity_ActivityType
            FOREIGN KEY (ActivityTypeID)
            REFERENCES ActivityType(ActivityTypeID)
```

Единственной принципиальной разницей между кодом, приведенным в листинге 3.12, и кодом из листинга 3.11 является наличие нескольких ограничений внешнего ключа. Обратите внимание, что ключевое слово ADD используется всего один раз, после чего через запятую указываются все необходимые ограничения.

На этом воплощение созданной ранее диаграммы отношений между объектами в базе данных SQLSpyNet можно считать полностью завершенным. В очередной раз вы проявили себя с наилучшей стороны, показав завидные для многих результаты. Поздравляю! Складывается впечатление, что вы самый одаренный из всех моих учеников!

Тем не менее, для того чтобы убедиться в работоспособности созданных отношений между таблицами (а также, чтобы подтвердить справедливость сказанных в ваш адрес комплиментов), в окне Enterprise Manager запустите уже знакомый вам редактор диаграмм.

Чтобы открыть созданную ранее диаграмму отношений между объектами (она носит имя *Current ERD*), дважды щелкните на соответствующей ей пиктограмме.

Щелкнув правой кнопкой мыши на области диаграммы и выбрав команду Add Table (Добавить таблицу), дополните текущую диаграмму созданными с помощью кода Transact-SQL таблицами, после чего упорядочите все восемь таблиц базы данных SQLSpyNet таким образом, чтобы они помещались в пределах одного окна.

Отметьте наличие всех созданных с помощью кода Transact-SQL отношений между таблицами. Окончательный вид диаграммы представлен на рис. 3.28.

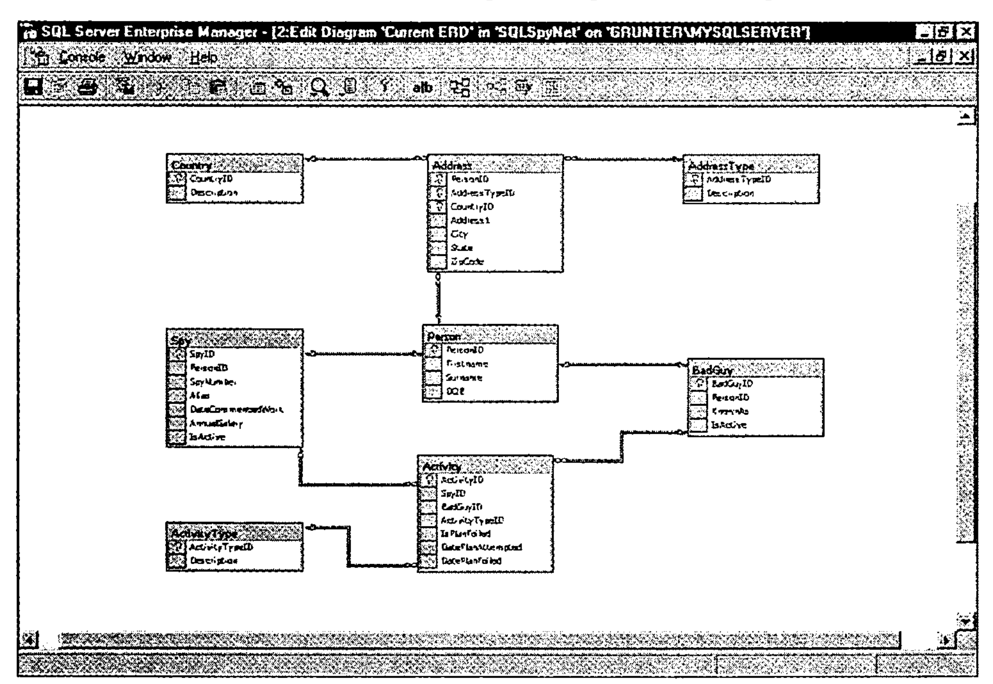


Рис. 3.28. Окончательный вид диаграммы отношений между объектами приложения SQLSpyNet

Обратите внимание на отношение между таблицами `Spy` и `Person`, а также между таблицами `BadGuy` и `Person`.

Пиктограмма с изображением ключа, расположенная по обоим концам упомянутых выше отношений, сигнализирует о типе зависимости между таблицами "один к одному". Без уникального индексирования столбца `PersonID` определение таких отношений было бы неверным.

Как вы, вероятно, уже отметили, сгенерированная с помощью SQL Server 2000 диаграмма отношения между объектами полностью совпадает с диаграммой отношения между объектами, окончательный вариант которой был представлен в начале данной главы. Это может обозначать только одно: мы полностью решили поставленную задачу.

Заполнение таблиц базы данных SQLSpyNet соответствующей информацией

В ходе освоения материала предыдущих разделов главы создана база данных SQLSpyNet и определены все ее базовые таблицы. Помимо этого, созданы отношения между базовыми таблицами, что обеспечивает поддержку целостности на уровне ссылок. Теперь осталось совсем немного: заполнить таблицы базы данных SQLSpyNet соответствующей информацией.

Вот тут-то вы и можете проявить свои творческие способности! Я покажу, как вводить в таблицу данные, и дам список значений, которые должны быть введены в каждую из таблиц. Помните, что эти списки являются всего лишь примерами данных, которые можно ввести в таблицы, так что вы вправе изменять их, добавляя или удаляя ту или иную информацию. Следует отметить, что приведенные в этом разделе списки данных содержат далеко не *всю* информацию, которая будет добавлена в таблицы базы данных SQLSpyNet; по мере продолжения работ по созданию приложения они будут пополняться все новыми и новыми данными.

Заполнение информацией следует начинать с таблиц самого верхнего уровня, или, как их еще называют, родительских таблиц. Причина использования такого подхода заключается в том, что для обеспечения целостности на уровне ссылок сначала следует убедиться, что соответствующие данные существуют в родительских таблицах, и лишь затем вносить их в дочерние таблицы. Действительно, если есть ребенок, то должен же где-то быть и его родитель!

Заполним данными таблицу `Country`. Для этого запустите программу Enterprise Manager и выберите таблицу `Country` в папке Tables базы данных SQLSpyNet.

Щелкнув правой кнопкой мыши, выберите из контекстного меню команду Open Table⇨Return all rows (Открыть таблицу⇨Возвратить все записи). Начиная с этого момента, мы будем называть подобное действие открытием таблицы (имя) в режиме просмотра данных. Например: откройте таблицу `Country` в режиме просмотра данных.

Открыв таблицу, щелкните на ее первой записи, которая должна представлять собой пустую строку. Поскольку первичный ключ таблицы `Country` является идентификационным столбцом (с автоматическим увеличением значения), SQL Server 2000 избавляет от необходимости вводить значение в поле `CountryID`. Более того, при попытке ввода информации в этот столбец SQL Server 2000 возвратит сообщение об ошибке. Как видите, вводить данные в идентификационные столбцы — одно удовольствие!

Выберите столбец `Description` и введите **United States**. Перейдите на следующую строку таблицы, непосредственно щелкнув на ней мышью или нажав клавишу Tab. В результате перехода на вторую строку столбцу `CountryID` первой строки таблицы должно быть автоматически присвоено значение 1, как показано на рис. 3.29.

Убедившись в том, что с автоматическим вводом значений в столбец `CountryID` все в порядке, заполните таблицу `Country` (столбец `Description`) приведенными ниже значениями.

Название страны
United States — уже введено
New Zealand
England
Ireland
Germany
Australia
Russia

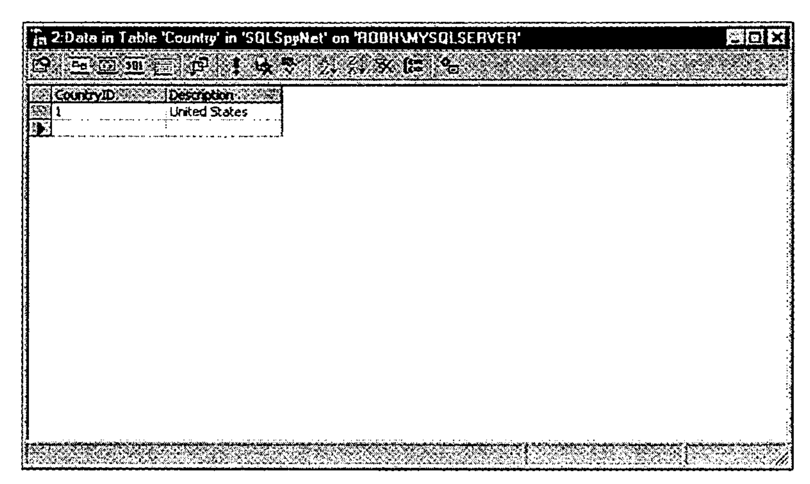


Рис. 3.29. SQL Server 2000 автоматически увеличивает значение идентификационного столбца `CountryID`

Приведенный список стран, естественно, невелик. При желании его можно расширить, хотя в процессе дальнейшей разработки приложения нам предстоит добавлять в него новые страны.

Перейдем к заполнению таблицы `AddressType`. Откройте ее в режиме просмотра данных. Поскольку таблица `AddressType` во многом аналогична таблице `Country` (имеет первичный ключ, являющийся идентификационным полем), введите приведенные ниже значения в поле `Description`.

Описание типа адреса
Business Postal address
Business Physical address
Home Postal address
Home Physical address

Аналогичные действия следует повторить и для таблицы `ActivityType`. Ниже приведен список значений для ввода в эту таблицу.

Описание типа деятельности
Take over the World!
World domination!
Chaos and mayhem
General disorder
Steal your favorite socks
Wreck havoc!

Что ж, стоит лишь отметить, что планы злоумышленников никогда не отличались особой оригинальностью!

Заполнив информацией все справочные таблицы, самое время проверить на работоспособность определенные ограничения целостности на уровне ссылок базы данных SQLSpyNet. Для этого попытаемся ввести адрес в таблицу `Address` без предварительного определения в таблице `Person` человека, которому соответствует этот адрес.

Помимо этого, проверим ограничение `NOT NULL`, определенное для столбца `PersonID` таблицы `Address`.

Напомним: при создании таблицы Address было решено, что ее столбец PersonID не может содержать значения NULL. Сначала проверим именно это ограничение таблицы Address, а затем перейдем к проверке целостности на уровне ссылок.

Откройте таблицу Address в режиме просмотра данных. Поскольку таблица Person пока что не содержит никакой информации, нельзя ввести значение в столбец PersonID. Оставив этот столбец незаполненным, тем самым позволим SQL Server 2000 ввести в него значение по умолчанию, которое в данном случае равно NULL.

Щелкните на столбце PersonID. Поскольку в таблице Person пока еще нет записи ни об одном человеке, нажмите клавишу <Tab> для того, чтобы перейти к столбцу AddressTypeID. При попытке сохранить эту запись в базе данных (это произойдет автоматически вследствие перехода на новую строку) SQL Server 2000 подставит в поле PersonID значение NULL.

Типы адресов хранятся в уже заполненной таблице AddressType. Согласно правилам целостности столбец AddressTypeID таблицы Address допускает ввод только тех значений, которые существуют в столбце AddressTypeID таблицы AddressType.

Пусть тип определяемого адреса будет *Home Physical Address* (Домашний адрес). В таблице AddressType соответствующая этому типу адреса строка имеет идентификационный номер 4 (напомним, что идентификационный номер определяется значением столбца AddressTypeID). Разумеется, в каждом конкретном случае значение столбца AddressTypeID может отличаться от 4, что связано с очередностью ввода типов адресов в таблицу.



Для того чтобы узнать идентификационный номер типа адреса *Home Physical Address*, откройте таблицу AddressType в режиме просмотра данных.

Введите значение в столбец CountryID. Пусть определяемый адрес принадлежит человеку из Новой Зеландии, а это значит, что потребуется узнать значение первичного ключа соответствующей записи в таблице Country. Эта строка имеет идентификационный номер 4, хотя в каждом конкретном случае данное число может варьироваться в зависимости от порядка ввода названия стран в таблицу.

Нажмите клавишу <Tab>, чтобы перейти к столбцу Address1, после чего введите в него значение **123 Hickory Street**. Снова нажмите клавишу <Tab> и введите в столбец City значение **Wellington**. Нажмите клавишу <Tab>. Поскольку административное деление Новой Зеландии не подразумевает наличия штатов, оставьте столбец State незаполненным.

Нажмите клавишу <Tab>, чтобы перейти к столбцу ZipCode. Почтовый индекс Веллингтона в Новой Зеландии — 6001; введите это значение в столбец ZipCode.

ZipCode — последний столбец таблицы Address. При переходе на следующую строку SQL Server 2000 автоматически попытается внести только что созданную запись в базу данных. При этом вначале будет выполнена проверка целостности информации, и в случае ее успешного завершения информация будет сохранена на диске. Для того чтобы сохранить на диске только что введенную информацию, перейдите на следующую строку таблицы Address.

В том случае, если вы следовали всем приведенным выше инструкциям, а также верно определили таблицу Address, SQL Server 2000 возвратит сообщение об ошибке, показанное на рис. 3.30.

Итак, проверка первой нестандартной ситуации прошла успешно, в результате чего вы убедились в работоспособности ограничения NOT NULL, определенного для столбца PersonID таблицы Address.

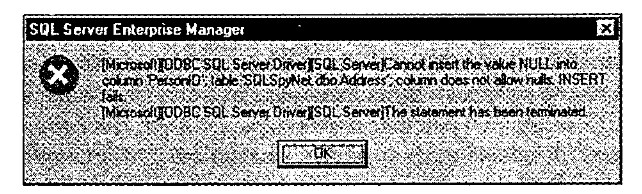


Рис. 3.30. SQL Server 2000 возвращает сообщение о нарушении ограничения NOT NULL

Щелкните на кнопке OK, после чего можно переходить к проверке правила целостности на уровне ссылок, определенного для таблиц Address и Person.

Щелкните на столбце PersonID и введите число **100**. Поскольку в таблице Person пока что не содержится ни одной записи, не имеет значения, какое число вы введете в столбец PersonID таблицы Address (в отличие, например, от столбца CountryID этой же таблицы). Попробуйте перейти к следующей строке таблицы Address.

SQL Server 2000 вновь осуществит попытку сохранить только что созданную запись в базе данных, однако из-за нарушения правила целостности на уровне ссылок (в таблице Person отсутствует запись с идентификационным номером 100) эта попытка завершится неудачей. Сгенерированное в результате сообщение об ошибке будет подобно приведенному на рис. 3.31.

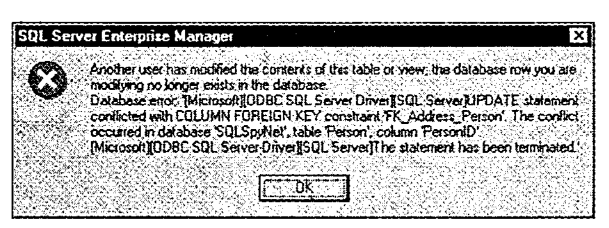


Рис. 3.31. SQL Server 2000 возвращает сообщение о нарушении ограничения целостности на уровне ссылок

Сообщение SQL Server 2000 о нарушении целостности на уровне ссылок информирует о попытке вставки в таблицу Address данных, противоречащих ограничению внешнего ключа этой таблицы (в данном случае ограничению FK_Address_Person). Часть сообщения *The statement has been terminated* указывает на то, что в конечном итоге информация не была сохранена в базе данных.

Как же избежать подобных проблем? Решение состоит в предварительном заполнении соответствующей информацией таблицы Person (по крайней мере, до заполнения зависящих от нее таблиц).

Откройте таблицу Person в режиме просмотра данных. Столбец PersonID является идентификационным столбцом данной таблицы, что подразумевает автоматический ввод его значений. Ниже приведены имена тайных агентов и злоумышленников, предназначенные для ввода в таблицу Person.

Имя	**Фамилия**	**Дата рождения**
James	Bondy	15 Jun 1963
Austin	Prowess	03 May 1942
Dr.	Eville	09 Sep 1941
Reilly	Ace of What?	10 Aug 1954
Barry	Bon Von Hausen	12 Jun 1972
Greg	Cross	04 Mar 1969
Claus	Von Schmit	31 Apr 1955

На заметку Обратите внимание на то, что все даты рождения приведены в таблице в так называемом среднем формате. Чаще всего используется формат даты *день/месяц/год*. Иногда встречаются серверы, на которых используется американский формат даты *месяц/день/год*, признаться, это всегда очень огорчало меня. Именно поэтому во избежание возможных недоразумений я решил привести все даты рождения в среднем формате. После ввода такой даты в поле таблицы SQL Server 2000 преобразует ее в короткий формат, так что в итоге все должны остаться довольны.

И снова хочу напомнить, что приведенный выше список тайных агентов и злоумышленников вовсе не претендует на полноту; он будет расширяться по мере дальнейшей разработки нашего приложения. Разумеется, это ни в коем случае не должно сдерживать ваш творческий порыв в стремлении заполнить таблицу `Person` любыми понравившимися именами.

Заполнив информацией основные таблицы базы данных SQLSpyNet, можно приступать к заполнению ее оставшихся таблиц — `Spy`, `BadGuy`, `Address` и `Activity`. Тем не менее отложим это до следующей главы, где вы узнаете, как заполнить перечисленные выше таблицы, используя код Transact-SQL.

Читая следующую главу, вы также сможете убедиться в гибкости предоставляемых SQL Server 2000 возможностей.

Наверняка вы отметили, что непосредственный ввод данных в таблицу — занятие довольно скучное и утомительное. Помимо этого, не так-то просто гарантировать корректность введенной информации, например выбор требуемых значений первичного ключа.

В следующей главе рассматривается несколько интересных операторов Transact-SQL, позволяющих проводить обновления связанной информации. Уверен, что вы уже вполне готовы к изучению "продвинутых" возможностей Transact-SQL! Однако сейчас подведем краткий итог проделанной работы.

Ах да, совсем забыл! Помните, я обещал вам рассказать в конце главы о транзитивной зависимости в таблице `Address`? Так вот, транзитивно зависимыми являются столбец `City` и, может быть, столбцы `State`, `ZipCode` и т.п. Все объекты, представленные в этих столбцах, независимы и могут быть выделены в различные справочные таблицы. Например, можно было бы создать таблицу `City`, зависящую от таблицы `Country` и связанную с таблицей `Address` через столбец `CityID`.

Почему же мы этого не сделали? Вспомните пословицу о том, что лучшее — враг хорошего. Иногда следует обращать внимание не только на преимущества, но и на недостатки высоконормализованной структуры. Для того чтобы получить чей-то адрес, понадобится объединить три или четыре таблицы, что негативно скажется на производительности базы данных. Таким образом, мы намеренно слегка денормализуем структуру SQLSpyNet ради обеспечения ее приемлемой производительности. (Признаюсь вам по секрету, что во время начального проектирования базы данных SQLSpyNet я не учел приведенных выше соображений и долго негодовал из-за ее медленной работы.)

Резюме

Итак, дамы и господа, настало время подвести итоги проделанной на текущий момент работы по созданию приложения SQLSpyNet. В ходе всего лишь одной главы создана база данных SQLSpyNet, спроектированы ее базовые таблицы и определены существующие между ними отношения.

Учитывая огромный объем этой главы, я не удивлюсь, если большинству читателей не удалось в полной мере запомнить изложенный в ней материал.

Несмотря на то что мы кратко затронули проблемы нормализации баз данных, настойчиво рекомендую вам углубить знания в этой области. Более глубокое понимание одной из важнейших концепций реляционной теории, которой является нормализация, оградит вас от множества разочарований и бессонных ночей в будущем.

И хотя созданная структура базы данных SQLSpyNet кажется вполне приемлемой, она все еще может быть подвергнута множеству различных улучшений. База данных никогда не бывает статическим объектом; она постоянно развивается в процессе разработки и уточнения требований и бизнес-правил. Не претендуя на то, что разработанная мною модель данных будет названа самой совершенной, не могу не отметить, что хорошая модель данных — это такая модель, которая при необходимости может быть легко адаптирована к новым требованиям.

Резюмируя сказанное выше, следует отметить, что главная цель при проектировании практически любого объекта заключается в создании надежного фундамента для продолжения дальнейшей работы. Хороший фундамент — залог того, что ваше здание будет оставаться крепким, невзирая на любые условия внешней среды.

Следующие шаги

В следующей главе продолжим изучение языка Transact-SQL, уделив особое внимание операторам управления данными (DML). Подробно будут рассмотрены операторы `SELECT`, `INSERT`, `UPDATE` и `DELETE`, а также некоторые весьма интересные операторы управления ходом выполнения программы.

Глава 4

Обработка данных с помощью кода Transact-SQL

В этой главе...

Эта глава полностью посвящена описанию операторов Transact-SQL. В предыдущей главе мы кратко затронули тему одного из подмножеств Transact-SQL — языка определения данных (Data Definition Language — DDL), более углубленному рассмотрению которого посвящена часть этой книги. Однако, прежде чем начать изучение глубинных аспектов языка определения данных, следует обратить внимание на другое подмножество Transact-SQL — язык управления данными (Data Manipulation Language — DML). Спросите, зачем? Как правило, большинство операторов языка определения данных основываются на использовании тех или иных операторов управления данными. Разумеется, есть и исключения; наиболее примечательное из них — рассмотренный ранее оператор CREATE TABLE. Большинство оставшихся операторов определения данных предназначены для извлечения, обработки или обновления содержащейся в базе данных информации.

Несмотря на то что в этой главе рассматриваются только самые распространенные операторы языка управления данными, оставшаяся часть книги изобилует множеством примеров различной сложности, каждый из которых обязательно должен быть выполнен в рамках создания приложения SQLSpyNet. А пока что рассмотрим основные операторы языка управления данными, которые пригодятся при обработке хранящейся в базе данных SQLSpyNet информации. Используя набор основных операторов, можно просмотреть, добавить, изменить и даже удалить из базы данных информацию о тайном агенте.

Следует отметить, что рассматриваемые операторы управления данными соответствуют стандарту ANSI SQL-92; это даст возможность использовать знания, полученные при освоении материала этой главы, при работе с любой СУБД, совместимой со стандартом ANSI SQL-92.

Так для чего же предназначен оператор SELECT?

Изучение операторов языка управления данными логичнее всего было бы начать с оператора SELECT. Во-первых, он уже упоминался в предыдущей главе при удалении базы

данных SQLSpyNet. Во-вторых, это один из наиболее часто используемых операторов Transact-SQL, и, кстати, именно к нему мы будем чаще всего обращаться в данной главе. Таким образом, изучение оператора SELECT имеет поистине стратегическое значение.

Оператор SELECT предназначен для извлечения информации из базы данных. Источником информации для этого оператора может быть отдельная таблица, группа связанных таблиц, таблица из другой базы данных и даже таблица, существующая на другом сервере! Следует оговориться, что во всех этих случаях необходимо иметь соответствующие права, позволяющие извлекать информацию из базы данных.

Благодаря своей структуре, во многом напоминающей структуру предложения в английском языке, SELECT стал одним из наиболее простых в изучении операторов языка Transact-SQL. Но не следует обманывать себя, считая этот оператор столь простым, что ему не стоит уделять особого внимания.

На самом деле оператор SELECT может быть записан настолько просто или настолько сложно, насколько того требует каждая конкретная ситуация, в которой он применяется. Оператор SELECT позволяет вкладывать другие операторы SELECT (подобная структура называется вложенным запросом), более того, он сам может быть вложен в операторы языка определения данных. Таким образом, когда упоминается оператор SELECT, автоматически подразумевается возможность его гибкого использования. Вопросы, касающиеся построения вложенных запросов, а также использования оператора SELECT совместно с операторами языка определения данных, будут рассматриваться более подробно в следующей главе, а пока что обратим свой взор на базовую структуру этого оператора.

Всего в данной главе рассматриваются четыре оператора языка управления данными: SELECT, INSERT, UPDATE и DELETE, используя которые, можно получить доступ к обработке практически любой информации, хранящейся в базе данных SQLSpyNet. Например, если кто-то из тайных агентов достигнет пенсионного возраста, можно отыскать соответствующую ему запись и обновить статус данного агента, изменив его с активного, скажем, на неактивный.

Термин

Согласно реляционной теории результатом выполнения оператора SELECT является так называемая *проекция* (*projection*) данных. Другими словами, информация, хранящаяся в таблице, "проектируется" в список результатов запроса. Полученная при выполнении запроса информация может представлять собой также сумму или усреднение хранящихся в таблице данных.

Оператор SELECT состоит из четырех основных "строительных" блоков.

- Собственно оператор SELECT, позволяющий извлекать из таблицы заданные столбцы информации.
- Предложение FROM, позволяющее задать таблицу (или таблицы), из которой будет проводиться выборка результатов запроса.
- Предложение WHERE, позволяющее определить ограничения, накладываемые на результат запроса.
- Предложение ORDER BY, позволяющее указать столбцы, по которым будет упорядочен результат запроса.

Ниже приведен базовый синтаксис оператора SELECT:

```
SELECT Имя_столбца1, Имя_столбца2, Имя_столбца3
FROM Имя_таблицы
WHERE Имя_столбца1 = ограничение
ORDER BY Имя_столбца1
```

Далее рассмотрим каждый компонент оператора SELECT отдельно, для того чтобы затем, собрав их вместе, составить запрос, который извлекал бы имя человека из таблицы Person. Рабочим инструментом при составлении запроса будет программа Query Analyzer, а целевой базой данных — SQLSpyNet (убедитесь в том, что она выбрана в качестве активной базы данных).



Для того чтобы сделать базу данных SQLSpyNet активной, необходимо выбрать ее из расположенного в строке меню программы *Query Analyzer* раскрывающегося списка или выполнить следующий оператор Transact-SQL:

Код для запуска

```
1: USE SQLSpyNet
2: GO
```

Первый строительный блок оператора SELECT

Как упоминалось ранее, оператор SELECT состоит из четырех основных блоков. В первой части этого оператора указывается список тех столбцов таблицы, данные которых будут рассматриваться в качестве объекта запроса. Именно эти данные (а в большинстве случаев только их часть) будут возвращены в качестве результата запроса.

Выберем в качестве объекта запроса таблицу Person. Обратите внимание, что синтаксис оператора SELECT не требует указания всех столбцов таблицы, как и не требует соблюдения порядка их перечисления (т.е. порядок перечисления столбцов может отличаться от порядка следования столбцов в таблице). Это свойство оператора SELECT позволяет легко подстраивать запрос под конкретные требования. Необходимо всего лишь выбрать нужные столбцы и перечислить их в произвольном (наиболее подходящем) порядке.

Итак, рассмотрим синтаксис первой части оператора SELECT.

Оператор выборки данных начинается с ключевого слова SELECT. За ним следует список разделенных между собой запятой (,) имен столбцов, которые необходимо извлечь из таблицы.

Одним из примеров использования оператора выборки является извлечение из таблицы Person имен всех людей, дни рождения которых приходятся на следующий месяц (предположим, необходимо разослать поздравительные открытки). Какая информация требуется в этом случае? Естественно, что наиболее логично было бы ограничиться именами и днями рождения. Таким образом, вместо того чтобы просматривать всю хранящуюся в таблице Person информацию, оператор SELECT позволяет ограничиться только самыми необходимыми ее столбцами.

Введите приведенный ниже код в окно ввода кода Transact-SQL программы Query Analyzer.

```
1: SELECT PersonID, DOB, Firstname, Surname
```

По мере изложения материала данный код будет дополнен еще несколькими фрагментами.



Некоторые разработчики используют символ звездочки (*) для того, чтобы указать необходимость извлечения информации из всех столбцов таблицы. Избегайте такой практики хотя бы потому, что это далеко не лучшее решение в подобной ситуации. Существует одна веская причина, по которой символ звездочки вообще не рекомендуется использовать при построении



оператора SELECT. Дело в том, что вся информация о базе данных SQL Server 2000, включая имена таблиц (системная таблица sysobjects) и имена столбцов (системная таблица syscolumns), хранится в скрытых системных таблицах. Таким образом, встретив символ звездочки в операторе SELECT, SQL Server 2000 должен будет просмотреть системные таблицы в поисках всех имен столбцов указанной таблицы. Можете себе представить, сколько времени займет подобная операция в базе данных больших размеров! Несмотря на то что ввод всех необходимых имен столбцов при построении запроса кажется довольно скучным занятием, выигрыш в производительности при увеличении объема базы данных с лихвой компенсирует все затраченные физические и моральные усилия. К тому же, имея такого помощника, как окно *Object Browser* программы *Query Analyzer*, можно в значительной мере облегчить себе эту тягостную задачу.

Обратите внимание на отсутствие запятой (,) после заключительного элемента в списке имен столбцов — столбца Surname. Именно таким образом в SQL указывается окончание перечисления имен столбцов в первой части оператора SELECT, описание которой, кстати, можно на этом считать завершенным. Что же дальше? Ах да, описание второй части оператора SELECT!

Как определить источник данных, или Описание предложения FROM

Вторая часть оператора SELECT представлена так называемым предложением FROM, которое предназначено для определения таблиц (или таблицы), служащих источником данных для запроса.

Для простоты ограничимся запросом к одной-единственной таблице. Разумеется, можно извлечь данные одновременно из таблиц Address, Person, AddressType и Country, получив таким образом всю информацию о конкретном человеке (в данном случае — его имя и адрес). Однако на текущий момент требуется только имя и дата рождения, т.е. та информация, которая хранится в одной таблице — Person. К тому же в главе 5, "Использование языка определения данных для просмотра и обновления информации", будет описано создание представления, предназначенного для извлечения всей информации о конкретном человеке. Итак, введите приведенный ниже код в окно ввода кода программы Query Analyzer, расположив его под строкой кода, рассмотренной в предыдущем разделе.

Код для запуска

```
1: SELECT PersonID, DOB, Firstname, Surname
2: FROM Person
```

Как видите, вся информация, которую необходимо указать в предложении FROM, представляет собой имена таблиц, являющихся источниками данных запроса.

Экскурс

Учитесь писать красиво!

Как вы наверняка уже отметили, в приведенном выше коде совсем не обязательно размещать предложение FROM на отдельной строке. Это было сделано с единственной целью — повысить читабельность кода, подчеркнув таким образом его деление на логические части. Правда, такой маленький фрагмент кода не так уж и сложен для восприятия, однако что бы

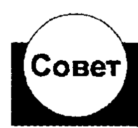


вы сказали о сегменте, логически состоящем из 15 строк и записанном всего лишь в одну?

По возможности старайтесь разбивать программный код на абзацы. Действуя таким образом, можно не только повысить его читабельность, но и облегчить поиск ответа на волнующий вопрос, разделив код на удобные, легко анализируемые фрагменты.

Хотя это и не обязательно (поскольку Transact-SQL является языком, не зависящим от регистра), я предпочитаю использовать при написании зарезервированных слов Transact-SQL только прописные символы (думаю, это уже вами подмечено), что, во-первых, позволяет выделить в коде ключевые слова Transact-SQL, а во-вторых, здорово помогает при его разборе.

Самое главное в том, что сейчас вы уже можете выполнить приведенный выше код. Оставшиеся две части оператора `SELECT` не обязательны и могут быть опущены. Итак, выполните приведенный выше код, щелкнув на пиктограмме с изображением зеленого треугольника.



Если вы предпочитаете выполнять команды путем нажатия соответствующих комбинаций клавиш, воспользуйтесь сочетанием <Alt+X> или <Ctrl+E>. Оба они предназначены для выполнения кода Transact-SQL, набранного в окне ввода кода программы *Query Analyzer*.

В случае корректного указания имен столбцов таблицы `Person` в области программы Query Analyzer, расположенной в нижней части окна ввода кода, должен появиться список всех людей, занесенных на текущем этапе в базу данных SQLSpyNet (рис. 4.1).

Рис. 4.1. Результат использования оператора `SELECT` для выборки информации из таблицы `Person`

Если в области, расположенной в нижней части окна ввода кода программы Query Analyzer, будет отсутствовать какой-либо подобный изображенному на рис. 4.1 результат (а вместо этого будет строка `0 row(s) affected` или, что еще хуже, сообщение об ошибке), сначала убедитесь в том, что в таблице `Person` содержится введенная в предыдущей главе информация. Получив сообщение об ошибке, например `Invalid column name` *'Имя_столбца'*, убедитесь в том, что на этапе ввода кода были корректно указаны все имена столбцов и таблиц.

Ограничение результатов запроса с помощью предложения WHERE

Приведенный выше код Transact-SQL имеет один существенный недостаток: он предназначен для использования только в том случае, когда необходимо извлечь все данные из таблиц (или таблицы). А что же делать, если нужно ограничиться информацией о каком-либо одном человеке? Вот тут-то и приходится вспомнить о третьей части оператора `SELECT` — предложении `WHERE`.

Предложение `WHERE` предназначено для сравнения данных путем использования таких логических операторов, как =, <, >, `LIKE` и т.п.

Рассмотрим структуру предложения `WHERE`. Она имеет следующий вид:
`WHERE` *имя_столбца логический_оператор некоторое_значение*

Введя приведенный ниже код в окно программы Query Analyzer, можно узнать имя, фамилию и дату рождения человека с идентификационным номером 3.

Код для запуска

```
1: SELECT PersonID, DOB, Firstname, Surname
2: FROM Person
3: WHERE PersonID = 3
```

Предложение `WHERE` вовсе не обязательно располагать на отдельной строке, но, следуя эстетическим соображениям...

Как и в предыдущем примере, приведенный выше код может быть незамедлительно выполнен. Результат его выполнения показан на рис. 4.2.

Обратите внимание, что в данном случае результат выполнения запроса состоит всего лишь из одной строки. Это демонстрирует эффективность применения предложения `WHERE`. Используя это предложение, можно достаточно гибко ограничивать объем возвращаемой в качестве результата запроса информации. Рассмотрим ситуацию, когда результат запроса будет состоять более чем из одной строки.

Изменим предложение `WHERE` таким образом, чтобы в результат выполнения запроса были включены записи только о тех людях, чьи фамилии начинаются на букву "B". Итак, удалите предложение `WHERE`, использовавшееся в предыдущем примере, и введите вместо него приведенный ниже код.

Код для запуска

```
1: SELECT PersonID, DOB, Firstname, Surname
2: FROM Person
3: WHERE Surname LIKE 'B%'
```

Выполните запрос и просмотрите полученный результат, обратив особое внимание на количество возвращенных в нем строк. Такая форма записи предложения `WHERE` позволяет расширить критерий поиска информации по сравнению с предыдущим применением предложения `WHERE`, оставаясь в рамках начальной формы запроса (определяемой оператором `SELECT` без предложения `WHERE`). Результат выполнения запроса представлен на рис. 4.3.

```
SELECT PersonID, DOB, Firstname, Surname
FROM Person
WHERE PersonID = 3
```

	PersonID	DOB	Firstname	Surname
1	3	1941-09-09 00:00:00.000	Dr.	Eville

Query batch completed | ZSOLTP\MYSQLSERVER (8.0) | sa (52) | SQLSpyNet | 0:00:00 | 1 rows | Ln 3, Col 19

Рис. 4.2. Результат выполнения оператора SELECT с использованием предложения WHERE

```
SELECT PersonID, DOB, Firstname, Surname
FROM Person
WHERE Surname LIKE 'B%'
```

	PersonID	DOB	Firstname	Surname
1	1	1963-06-15 00:00:00.000	James	Bondy
2	5	1972-06-12 00:00:00.000	Barry	Bon Von Hausen

Query batch completed | ZSOLTP\MYSQLSERVER (8.0) | sa (52) | SQLSpyNet | 0:00:00 | 2 rows | Ln 3, Col 1

Рис. 4.3. Результат выполнения оператора SELECT с использованием измененной формы предложения WHERE

На заметку

Небольшое замечание по поводу только что использованной формы предложения WHERE. Знак процента (%) указывает на проведение так называемого *поиска по шаблону*. Согласно данному шаблону в качестве результата запроса будут возвращены записи, соответствующие тем людям, чьи фамилии начинаются на букву "B", например Black или Brown. Для того чтобы извлечь из таблицы записи, соответствующие людям, в фамилиях которых только *встречается* буква "B", необходимо поместить знак процента по обе стороны от символа "B", например %B%.

Следует отметить, что одному оператору SELECT может соответствовать несколько критериев WHERE. Например, для того чтобы задать два критерия поиска информации, необходимо ввести первый критерий и отделить его от второго критерия ключевым словом AND или OR.



Символ "B", использовавшийся в предыдущем примере, зависит от регистра, поскольку того требует языковая настройка, выбранная на этапе создания базы данных. Напомню, что при создании базы данных мы использовали языковую настройку SQL Server 2000 по умолчанию, которая, например, в данном случае предполагает зависимость от регистра. Следует отметить, что языковая настройка SQL Server 2000 полностью зависит от параметров, выбранных на этапе установки этой СУБД.

Для проверки языковой настройки базы данных используется программа *Enterprise Manager*. Щелкните на соответствующей базе данных SQLSpyNet пиктограмме, расположенной в папке *Databases* (Базы данных). Выбрав SQLSpyNet, щелкните на ней правой кнопкой мыши и выполните команду *Properties* (Свойства). Во вкладке *General* (Общие) диалогового окна свойств находится параметр *Collation name* (Название языковой настройки). Значение этого параметра соответствует языковой настройке, выбранной при создании базы данных. (Отсутствие значения параметра *Collation name* предполагает использование языковой настройки сервера по умолчанию.)

Обратите внимание: *Collation name* является параметром, доступным только для чтения. Это объясняется тем, что смена языковой настройки приведет к переупорядочению и переиндексированию всей хранящейся в базе данных информации. К сожалению, для того чтобы изменить языковую настройку, базу данных необходимо сначала удалить, а затем создать заново.

Экскурс

Сужение области поиска данных с помощью добавления критериев

Ключевые слова AND и OR позволяют задать одновременно несколько критериев, ограничивающих область поиска данных. Грубо говоря, ключевое слово AND подразумевает необходимость использования "этого критерия *и* этого критерия". Ограничение, накладываемое словом AND, несколько строже ограничения, накладываемого словом OR, которое подразумевает необходимость использования "этого критерия *или* этого критерия".

В большинстве случаев для ограничения области поиска данных используется ключевое слово AND, имеющее более высокий приоритет, чем OR. Так, в приведенном ниже примере первым будет вычислено ограничение, заданное с помощью ключевого слова AND.

```
AND Surname LIKE '%B%'
AND DOB < '15 Jul 1980'
OR Firstname LIKE '%j%'
```

Экскурс

Таким образом, приведенное выше ограничение будет интерпретировано как "критерий Surname AND критерий DOB OR последний критерий".

В результате выполнения запроса с подобным ограничением SQL Server 2000 возвратит информацию обо всех людях, в фамилиях которых присутствует символ "B" и которые родились до 15 июля 1980 года, а также обо всех, в чьих именах присутствует символ "j".

В школьные годы вы наверняка проходили порядок выполнения операций в математике: сначала идут скобки, затем возведение в степень, деление, умножение, сложение и, наконец, вычитание. Такой же порядок операций существует и в Transact-SQL. Каждый оператор имеет свой порядок выполнения, что, естественно, приводит к наличию более высокого приоритета у некоторых операторов, это было показано на примере операторов AND и OR. Таким образом, я был прав, отмечая в самом начале книги, что вам еще не раз пригодятся полученные в школе знания!

Постройте этих шпионов по росту, или Использование предложения ORDER BY

Рассмотрим четвертую, заключительную часть оператора SELECT — предложение ORDER BY. Это предложение позволяет задать порядок следования столбцов в возвращаемом результате запроса, упорядочив их относительно одного или нескольких указанных столбцов. Обратите внимание на порядок следования столбцов в результате выполнения запроса, полученном в предыдущем примере. Для того чтобы изменить этот порядок, следует добавить к существующему оператору SELECT предложение ORDER BY. В окне программы Query Analyzer введите четвертую строку приведенного в листинге 4.1 кода, расположив ее непосредственно после предложения WHERE.

Листинг 4.1. Сортировка результата выполнения запроса по полю *Firstname*

Код для запуска

```
1: SELECT PersonID, DOB, Firstname, Surname
2: FROM Person
3: WHERE Surname LIKE '%B'
4: ORDER BY Firstname
```

Выполните приведенный код. Отметьте, что в данном случае результат выполнения запроса был упорядочен по возрастанию относительно столбца Firstname. Здорово, не правда ли?

Подобно оператору SELECT, предложение ORDER BY допускает указание списка разделенных между собой запятыми столбцов, по которым должен быть упорядочен результат выполнения запроса. Помимо этого, существует возможность непосредственного определения способа упорядочения результатов запроса. Для упорядочения по возрастанию используется (по умолчанию) ключевое слово ASC, а для упорядочения по убыванию — ключевое слово DESC. Чтобы продемонстрировать все упомянутые свойства предложения ORDER BY, внесем небольшие коррективы в код, приведенный в листинге 4.1.

Итак, изменим рассмотренное выше предложение ORDER BY, как показано в листинге 4.2.

Листинг 4.2. Сортировка результата запроса в алфавитном порядке

Код для запуска ➔

```
1: SELECT PersonID, DOB, Firstname, Surname
2: FROM Person
3: WHERE Surname LIKE '%B'
4: ORDER BY Firstname DESC, DOB
```

SQL Server 2000 отсортирует результат выполнения запроса в порядке убывания сначала по столбцу Firstname, а затем по столбцу DOB. Таким образом, если в результате выполнения запроса встретятся две записи с одинаковыми фамилиями *Harry*, то запись, соответствующая более "старому" *Harry*, будет идти первой.

Вот и все! Теперь у вас есть полное представление о базовой структуре оператора SELECT. В процессе последующей разработки приложения продолжим знакомство с этим оператором, изучив вложенные запросы, внутреннее, левое и правое соединения, а также многое другое.

Как вы уже наверняка догадались, SELECT — чрезвычайно мощный оператор Transact-SQL. С его помощью можно проверить существование информации в базе данных, возвратить требуемый набор данных ее пользователям, а также сгенерировать отчет для анализа на "высшем" уровне управления базой данных.

Внесение информации в базу данных с помощью оператора INSERT

Рассмотрев оператор, с помощью которого извлекается информация из базы данных, самое время познакомиться со средствами Transact-SQL, выполняющими обратную операцию, т.е. внесение информации в базу данных. Как следует из названия этого раздела, оператором, предназначенным для внесения данных, является INSERT.

Оператор INSERT позволяет добавлять новые значения в таблицы базы данных. С его помощью можно внести в базу данных не только несколько строк информации, но и с использованием оператора SELECT даже целую таблицу, заполненную данными! Следует отметить, что выполнение последней операции требует, как правило, наличия определенных прав и соблюдения правил целостности.

Структура оператора INSERT несколько отличается от структуры оператора SELECT, хотя в основном они базируются на одних и тех же принципах.

Оператор INSERT состоит из двух основных частей.

- Оператора INSERT INTO позволяет указать таблицу и ее столбцы, в которые будет вноситься информация.
- Оператора VALUES позволяет указать значения, вносимые в таблицу. Между количеством указанных значений и количеством столбцов таблицы должно существовать взаимно-однозначное соответствие; другими словами, они обязательно должны совпадать.

Лучший способ изучения оператора INSERT — рассмотрение практических примеров его использования. В качестве целевой таблицы выберем, как и при изучении оператора SELECT, таблицу Person. На самом деле, пока что катастрофически не хватает как тайных агентов, так и злоумышленников, поэтому использование оператора INSERT для внесения новых "темных личностей" в базу данных будет весьма кстати.

Определение таблицы и столбцов, в которые необходимо внести новую информацию

Сначала введите следующий код в окно ввода кода программы Query Analyzer (убедитесь, что в качестве активной выбрана база данных SQLSpyNet):

```
1: INSERT INTO Person (Firstname, Surname, DOB)
```

В отличие от оператора `SELECT`, приведенный выше код еще не готов к выполнению. Прежде чем дополнить этот код недостающим оператором `VALUES`, обратим внимание на структуру оператора `INSERT INTO`.

Оператор `INSERT INTO` указывает SQL Server 2000 тип выполняемой команды Transact-SQL. Непосредственно после ключевых слов `INSERT INTO` следует ввести имя таблицы, а затем разделенный запятыми список столбцов этой таблицы, в которые будет внесена новая информация.

На заметку При внесении информации во все столбцы таблицы синтаксис языка Transact-SQL позволяет не перечислять их имена, что, однако же, не избавляет от ответственности за ввод всех соответствующих этим столбцам значений.

Обратите внимание на отсутствие в списке столбцов таблицы `Person` ее первичного ключа — столбца `PersonID`. Это было сделано намеренно, так как при создании таблицы `Person` столбец `PersonID` был объявлен ее идентификационным столбцом, а SQL Server 2000 запрещает неавтоматический ввод значений в столбец такого типа.

Поскольку соответствующее значение идентификационного столбца будет подставлено SQL Server 2000 автоматически, его можно не включать в список столбцов таблицы `Person`, в которые будет внесена новая информация.

На заметку Стоит отметить, что способ, которым SQL Server 2000 обрабатывает идентификационный столбец таблицы, может быть изменен, однако пока что не станем вносить коррективы в эту область функционирования СУБД.

Внесение информации в таблицу

Вторая часть оператора `INSERT` предназначена для указания фактических значений, которые будут внесены в таблицу. Введите приведенный ниже код в окно программы Query Analyzer.

Код для запуска

```
1: INSERT INTO Person (Firstname, Surname, DOB)
2: VALUES ('Margaretha', 'Zelle', '7 Aug 1876')
```

Обратите внимание на то, что все вносимые в таблицу значения указываются в кавычках ('). Заключения в кавычки требуют практически все типы данных, кроме числовых (целые числа, денежные суммы, числа с плавающей запятой и т.д.). Существует эмпирическое правило, гласящее, что если вносимое в таблицу значение представляет собой строку или содержит по крайней мере один символ, отличный от цифры (включая символ пробела), то это значение следует взять в кавычки. Однако даже такое правило имеет несколько небольших исключений. Например, если вместо значения указывается ключевое слово `NULL`, то его, вопреки правилу, не следует брать в кавычки.

В кавычки следует заключать даже дату (как показано в приведенном выше коде). Спросите, почему? Большинство дат, помимо цифр, включают символы прямого слеша (/) или символы пробела для разделения составляющих их значений, а потому, несмотря на то что они начинаются с цифры, содержат строковые символы и таким образом подпадают под только что сформулированное правило.

Выполните приведенный выше код. Если операция внесения данных в таблицу завершится успешно, SQL Server 2000 с помощью сообщения подтвердит внесение в таблицу одной строки информации, как показано на рис. 4.4.

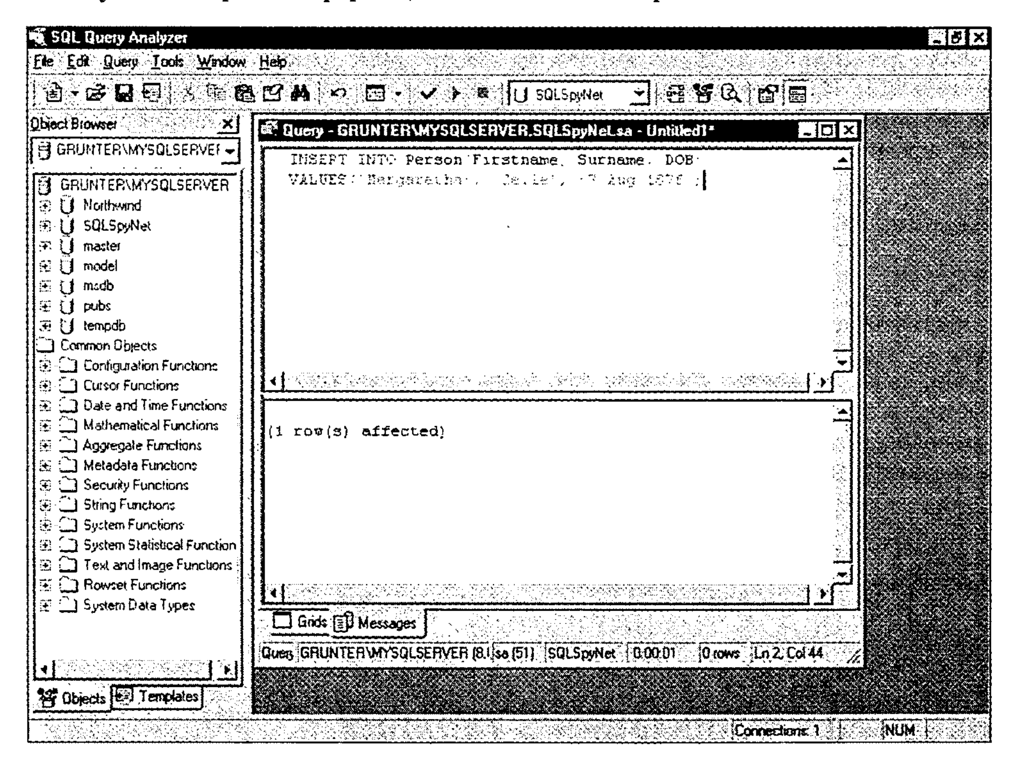


Рис. 4.4. SQL Server 2000 подтверждает успешное выполнение оператора `INSERT`

Воспользуемся оператором `INSERT` для того, чтобы внести в таблицу `Person` еще одну новую запись. Введите следующий код в окно программы Query Analyzer:

Код для запуска

```
INSERT INTO Person (Firstname, Surname)
VALUES ('Emmat', 'Peels')
```

Обратите внимание, что на этот раз значение для столбца `DOB` не указывается. При внесении такой строки в таблицу `Person` SQL Server 2000 автоматически подставит в столбец `DOB` значение `NULL`.

После выполнения приведенных выше операторов `INSERT` необходимо убедиться в том, что соответствующая информация действительно была внесена в базу данных. Для этого воспользуемся рассмотренным ранее оператором `SELECT`, введя в окно программы Query Analyzer следующий код:




```
SELECT * FROM Person ORDER BY PersonID
```

Результатом выполнения данного кода будет список всех строк таблицы Person, упорядоченный по столбцу PersonID.



> В предыдущем примере мы использовали символ звездочки (*) для указания необходимости извлечения всех столбцов таблицы Person. Как отмечалось ранее, это далеко не самое удачное решение, которого рекомендуется избегать при разработке реальных приложений, взаимодействующих с базой данных. На самом деле подобную практику не следовало бы использовать и в рассматриваемом примере, так как выполнение приведенного выше оператора SELECT неизбежно скажется на производительности SQL Server 2000.
>
> Единственной причиной, по которой символ звездочки был использован в предыдущем примере, является желание продемонстрировать различные способы выполнения оператора SELECT. Но хочу еще раз предупредить, что подобная практика может привести к значительному ухудшению производительности СУБД, так что по возможности ее следует избегать.

Помимо информации, внесенной при создании базы данных Person, список всех ее строк будет включать также две новые строки. Поскольку эти строки были внесены в таблицу самыми последними, они будут отображены в конце списка.

Аналогично тому как это делалось при использовании программы Enterprise Manager, SQL Server 2000 проверяет соблюдение всех ограничений базы данных при добавлении информации с помощью кода Transact-SQL. Вследствие этого попытка выполнения приведенного ниже кода завершится неудачно (информация не будет внесена в таблицу), как показано на рис. 4.5.

Код для запуска

```
INSERT INTO Person (Firstname, Surname, DOB)
VALUES (NULL, 'Someone', '23 Dec 1977')
```

SQL Server 2000 возвратит сообщение об ошибке, в котором будут подробно объяснены причины неудачного выполнения оператора INSERT.

Как видите, оператор INSERT предоставляет достаточную гибкость при внесении информации в базу данных. Однако здесь существует принципиальная проблема: как правило, большинство пользователей не знают синтаксиса оператора INSERT, а поэтому не смогут самостоятельно внести информацию в базу данных. Что же делать в подобной ситуации? Терпение и еще раз терпение. Скоро вы сами обо всем догадаетесь.

Ну а если серьезно, то в процессе дальнейшей разработки приложения вы научитесь скрывать подобные кажущиеся сложными операторы от пользователей системы и сможете предоставить им достаточно легкий способ обновления информации в базе данных. Это будет достигнуто с помощью так называемых хранимых процедур, рассматриваемых в следующей главе.

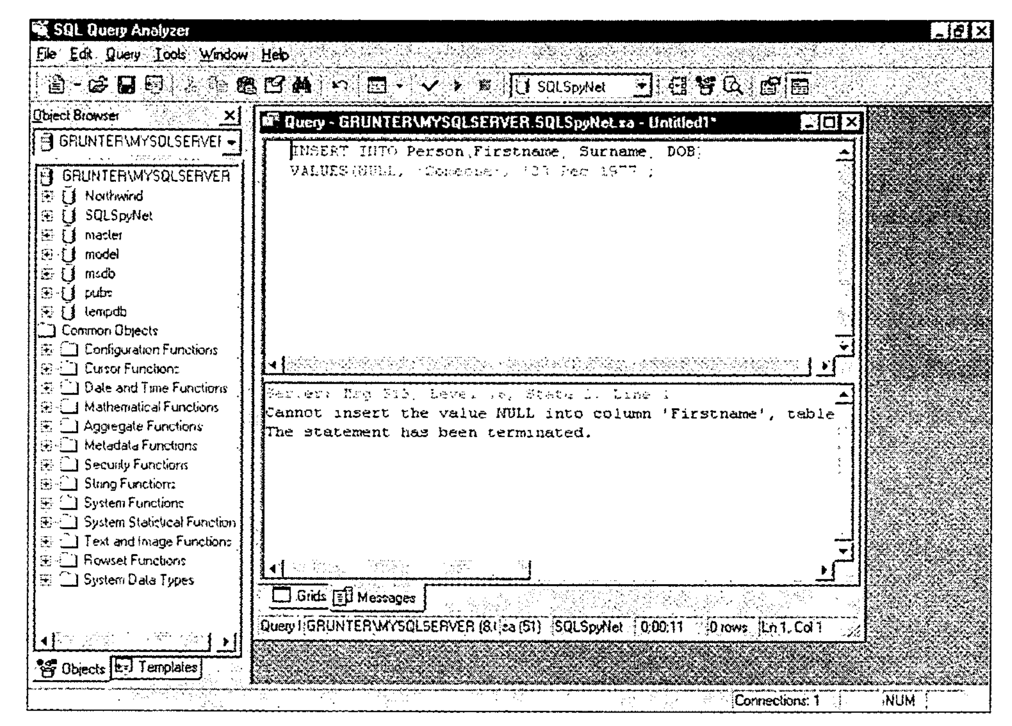


Рис. 4.5. Поскольку столбец `Firstname` таблицы `Person` имеет ограничение `NOT NULL`, выполнение приведенного выше оператора `INSERT` завершится сообщением об ошибке

Обновление информации в базе данных

Обновление информации в базе данных осуществляется с помощью оператора UPDATE. Как вы уже наверняка догадались, этот оператор позволяет изменять значения одного или нескольких столбцов и строк таблицы. Структура оператора UPDATE во многом напоминает структуру уже рассмотренных ранее операторов, что способствует его достаточно легкому изучению и применению на практике. Аналогично предыдущим операторам, уровень сложности UPDATE зависит от его конкретного употребления, т.е. можно создать как очень простой, так и очень сложный по своей структуре оператор.

Одной из наиболее примечательных особенностей оператора UPDATE является возможность обновления сразу нескольких столбцов в строке (или строках) таблицы.

Следует отметить, что оператор UPDATE имеет строго целевое назначение, т.е. его можно использовать только для обновления информации в таблице (а не для добавления или удаления строк).

Начиная с момента заполнения базы данных информацией, UPDATE наверняка займет абсолютное первое место по частоте использования (а также, к сожалению, и по частоте ошибок) среди других операторов Transact-SQL за исключением разве что оператора SELECT.

Оператор UPDATE состоит из трех частей: названия таблицы, в которой необходимо изменить информацию, разделенного запятыми списка изменяемых столбцов и новых значений, а также критерия WHERE (по усмотрению).

Рассмотрим пример использования оператора UPDATE.

Определение целевой таблицы

При добавлении информации в таблицу `Person` мы намеренно допустили одну небольшую орфографическую ошибку. К счастью, в Transact-SQL есть такой оператор, как `UPDATE`, позволяющий буквально “на лету” изменять нужные данные. Однако, прежде чем воспользоваться оператором `UPDATE`, убедимся с помощью оператора `SELECT` в том, что таблица `Person` содержит несколько новых строк информации, внесенных в нее с помощью оператора вставки `INSERT`.

Введите приведенные ниже строки кода в окно ввода кода программы Query Analyzer.

```
SELECT PersonID, Firstname, Surname, DOB
FROM Person
WHERE PersonID = 8
```



Выполняйте оператор выбора данных `SELECT` каждый раз перед внесением изменений в таблицу и после этого.

Для того чтобы максимально упростить эту задачу, следует открыть новое окно ввода кода программы *Query Analyzer* и набрать в нем нужный оператор `SELECT`, после чего к этому окну можно возвращаться всякий раз при необходимости убедиться в корректном выполнении всех изменений.

Итак, создадим первый запрос с использованием оператора `UPDATE`. Откройте новое окно ввода кода, для чего выполните команду File⇨New (Файл⇨Новый).



Эту же операцию можно выполнить, использовав комбинацию клавиш <Ctrl+N>.

Введите следующий код:

```
UPDATE Person
```

Это всего лишь первая часть оператора `UPDATE`, которая, как можно было легко догадаться, указывает SQL Server 2000 на необходимость изменения информации в таблице. Непосредственно после ключевого слова `UPDATE` указывается имя обновляемой таблицы. Будьте терпеливы, поскольку, аналогично оператору `INSERT`, приведенный выше код еще не готов к выполнению.

Внесение изменений в некорректно введенные данные таблицы

После указания имени таблицы следует указать также имя ее обновляемого столбца (или столбцов). Введите приведенный ниже код в окно ввода кода программы Query Analyzer.

```
UPDATE Person
SET Firstname = 'Emma',
Surname = 'Peel'
```

Этот код указывает SQL Server 2000 на необходимость установки значения *Emma* столбца `Firstname` в значение *Peel* столбца `Surname`. В случае обновления только одного столбца таблицы в приведенном выше коде следовало бы избавиться от запятой (`,`) и от всего, что стоит после нее.



Ни в коем случае не выполняйте данный код, несмотря на то что он в принципе уже вполне пригоден к выполнению! Причина такого строгого запрета заключается в том, что смысл приведенного выше кода — обновление столбцов `Firstname` и `Surname` абсолютно во *всех* строках таблицы `Person`.

Эта особенность оператора `UPDATE` была разработана намеренно и в данный момент поддерживается всеми системами управления реляционными базами данных (СУРБД). Причина, по которой это было сделано, заключается в том, что время от времени возникают ситуации, требующие обновления всех строк заданной таблицы. Примером такого обновления может быть увеличение заработной платы всех служащих некоторой организации на 15%.

Определение строк таблицы, требующих обновления

Каким же образом можно обновить информацию только в тех строках таблицы, где это необходимо? Здесь на помощь приходит критерий `WHERE`, аналогичный одноименному критерию оператора `SELECT`.

Для того чтобы гарантировать обновление только одной строки таблицы, следует воспользоваться ее первичным ключом. Напомним, что первичный ключ обеспечивает уникальность хранящейся в таблице информации. Таким образом, определенному значению первичного ключа соответствует только одна строка таблицы. Для того чтобы завершить формирование оператора `UPDATE`, введите код из листинга 4.3.

Листинг 4.3. Использование первичного ключа таблицы для обновления информации в заданной строке

Код для запуска

```
UPDATE Person
SET Firstname = 'Emma',
    Surname = 'Peel'
WHERE PersonID = 8
```

Вот теперь-то можно с чистой совестью выполнить оператор `UPDATE`. В результате SQL Server 2000 возвратит сообщение об изменении информации в одной строке таблицы, подобное представленному на рис. 4.6.

На заметку Более подробная информация о критерии `WHERE` содержится в описании оператора `SELECT` ранее в этой главе.

Выполните оператор выборки данных `SELECT`. Обратите внимание на изменение информации в строке со значением первичного ключа `PersonID`, равным 8. На этом обсуждение базовой структуры оператора `UPDATE` можно считать завершенным.

Какое же значение имеет оператор `UPDATE` для разрабатываемого приложения SQLSpyNet? Поистине огромное, так как после заполнения базы данных соответствующей информацией это будет один из наиболее часто используемых операторов, выполняемых вами и вашими сотрудниками практически регулярно.

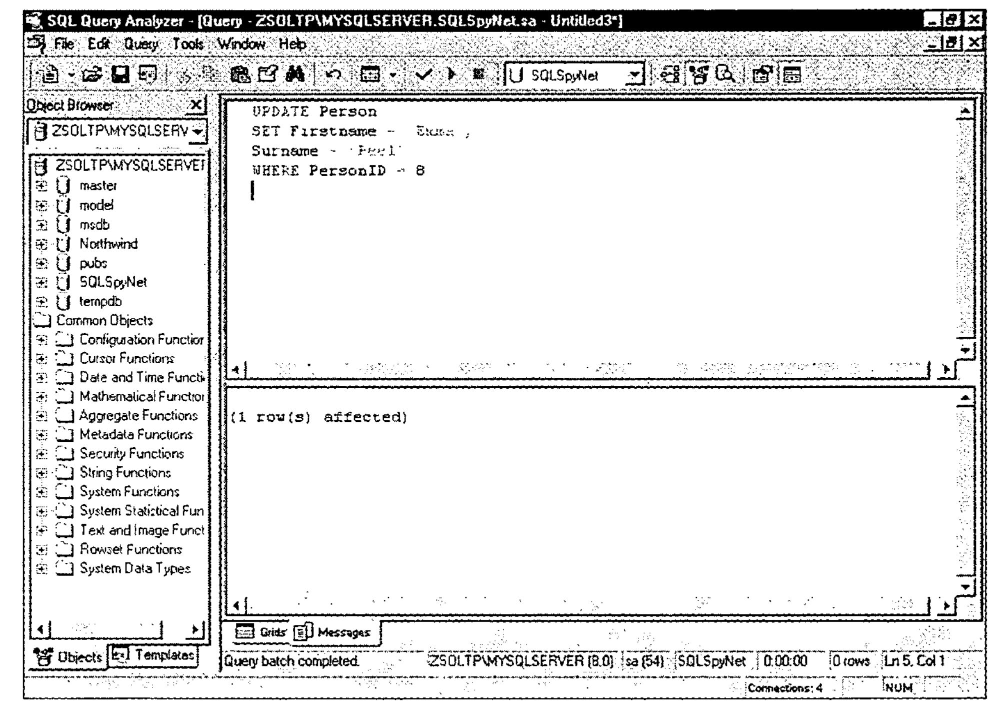


Рис. 4.6. Сообщение об успешном выполнении оператора UPDATE

Только представьте себе, что каждый раз после завершения очередной операции вам придется обновлять таблицу Activities, внося в нее (с радостью или с огорчением) информацию об успешном или провальном завершении деятельности тайного агента! К тому же новые сведения о злоумышленниках по мере поступления должны заноситься в соответствующую таблицу для того, чтобы тайный агент мог всегда иметь под рукой самую свежую информацию. Также оператор UPDATE будет часто использоваться внутри некоторых операторов определения данных, изучение которых, как говорится, уже не за горами.

Окончание пути тайного агента — оператор DELETE

В предыдущих разделах этой главы рассматривались вопросы извлечения, вставки и обновления информации в базе данных. Однако что же делать в том случае, если некоторая часть хранящейся в таблице информации будет больше не пригодна для использования (безнадежно устареет)? Здесь на помощь приходит последний из рассматриваемых в этой главе операторов управления данными — DELETE. Оператор DELETE очень мощный, он позволяет удалить из базы данных всю связанную с определенной записью информацию в соответствии с существующими ограничениями. Его структура во многом напоминает структуру рассмотренного ранее оператора SELECT. В зависимости от спецификации оператора DELETE, из базы данных может быть удалена вся, часть или всего лишь одна строка информации.



Будьте осторожны, так как с помощью оператора `DELETE` можно буквально "в мгновение ока" удалить все строки таблицы.

Некоторые читатели могут отнестись пренебрежительно к приведенному выше замечанию, посчитав, что таблица с каким-то жалким десятком записей не стоит такой заботы. Это их право, мое дело — всего лишь рассказать вам об использовании оператора `DELETE`, а все, что случится после этого, уже не входит в пределы моей компетенции. Скажу только, что в недалеком будущем многим из вас придется иметь дело с гораздо большими объемами информации, случайная потеря хоть части которой может стоить рабочего места. Другими словами, пока вы не будете абсолютно уверены в том, что у порога вашей двери не стоит очередной злоумышленник, проявляйте максимальную бдительность при удалении какой-либо информации из базы данных.

Прежде чем перейти к практическому примеру, откройте новое окно ввода кода программы Query Analyzer и введите в нем следующий оператор `SELECT`, предназначенный для проверки результатов выполнения оператора `DELETE`:

Код для запуска

```
SELECT PersonID, Firstname, Surname, DOB
FROM Person
```

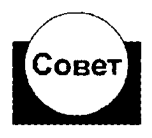
Приступим к практической части изучения оператора `DELETE`. Введите следующий код в окно ввода кода программы Query Analyzer:

```
DELETE FROM Person
```



Подобно оператору `UPDATE`, приведенный код уже готов к компиляции и выполнению. Тем не менее ни в коем случае НЕ ВЫПОЛНЯЙТЕ его на данном этапе, так как это приведет к удалению всей информации из таблицы `Person`. Для того чтобы ограничить количество удаляемых строк, следует воспользоваться критерием `WHERE`, который рассматривался ранее в этой главе.

Обратите внимание на то, что, в отличие от всех других операторов управления данными, в приведенном выше примере отсутствует какая-либо информация, касающаяся столбцов таблицы `Person`. Это объясняется тем, что наименьшим элементом таблицы, который может быть удален, является строка (а не столбец).



Для того чтобы удалить значение отдельного поля таблицы, следует воспользоваться оператором `UPDATE`, присвоив этому полю значение `NULL`.

Введите и выполните следующий код Transact-SQL:

Код для запуска

```
DELETE FROM Person
WHERE PersonID = 6
```

Обратите внимание на способ использования первичного ключа таблицы `Person`; это гарантирует "хирургически точное" удаление нужной нам строки таблицы.

После выполнения кода SQL Server 2000 возвратит сообщение, подтверждающее внесение в таблицу соответствующих изменений, как показано на рис. 4.7.

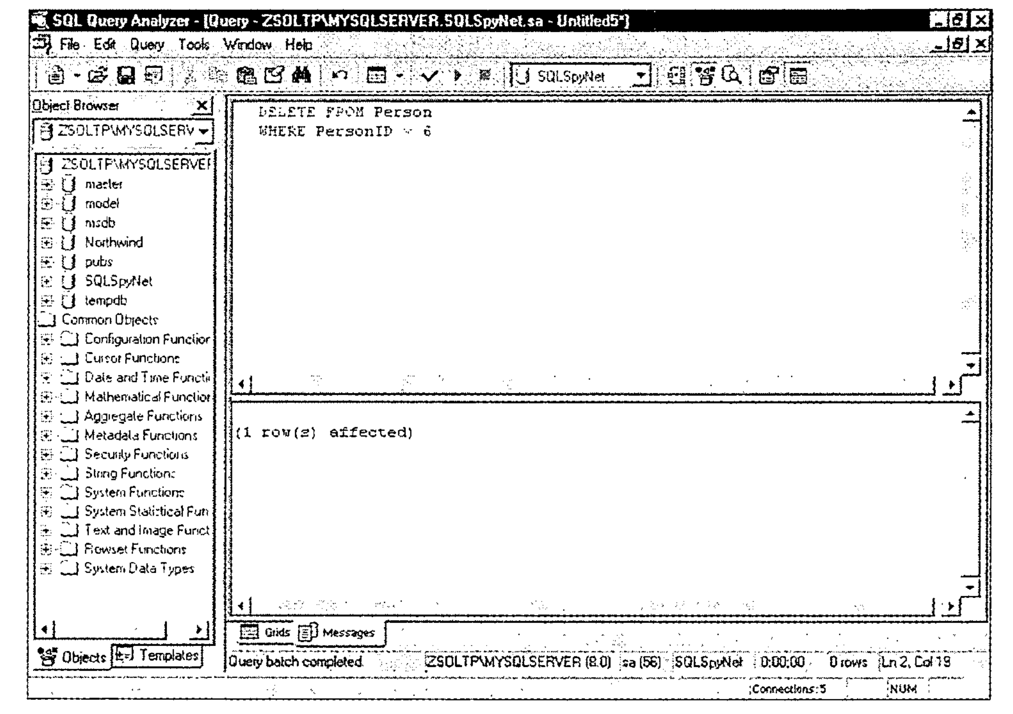


Рис. 4.7. Результат использования оператора `DELETE` *для удаления информации из таблицы* `Person`

А теперь выполните повторно оператор `SELECT`, приведенный в начале данного раздела. На этот раз в качестве результата будет возвращено на одну строку меньше, чем при выполнении этого же оператора перед удалением строки из таблицы.

Итак, "легким движением руки" мы избавились от строки таблицы `Person` со значением поля `PersonID`, равным 6. Магия? Нет, самая что ни на есть реальность!

Подведем итог. Только что мы завершили изучение основных операторов языка управления данными. (Только не говорите, что это было слишком сложно!) Можно заметить, что все операторы имеют похожую структуру, что довольно существенно облегчает процесс их изучения. Разобравшись с предназначением каждого из операторов управления данными, вы заложите крепкий фундамент, который позволит выполнять практически любые операции, связанные с обработкой информации в базе данных.

Не беспокойтесь об отсутствии практического опыта — по мере изучения материала этой книги (а также по мере развития вашей карьеры) через ваши руки пройдет множество различных операторов управления данными. Ну а пока что давайте перейдем к следующему разделу, который озаглавлен...

Чем богат Transact-SQL, помимо уже описанных операторов?

Помните, ранее в этой книге я назвал Transact-SQL языком множеств, а не языком программирования? В наши дни это утверждение выглядит не совсем корректно. Объяснить, почему? Действительно, не изменять же своим старым привычкам!

Дело в том, что стандарт ANSI SQL-92 был расширен в большинстве версий SQL Server (а также в других СУРБД) добавлением некоторых операторов, свойственных, прежде всего, языкам программирования.

На сегодняшний день сложилась такая ситуация, при которой база данных уже не может ограничиться выполнением только основных задач с использованием для этого операторов определения данных и операторов управления данными. Понимая необходимость повысить гибкость средств управления базой данных, Microsoft разработала так называемые *управляющие операторы*. Они выполняют действия, подобные действиям эквивалентных операторов в языках программирования. Это подразумевает возможность объявления переменных, использования операторов `IF` и `WHILE`, выдачи сообщений пользователям с помощью оператора `PRINT`, а также множество других функций.

Кроме повышения функциональности операторов определения и операторов управления данными, управляющие операторы позволяют добиться большей гибкости при выполнении различных операций над хранящейся в базе данных информацией. Например, перед обновлением данных отнюдь не лишне убедиться в самом факте существования этих данных. Также очень часто требуется определить влияние внесенных в таблицу изменений на другие таблицы базы данных. Несмотря на то что приведенные примеры могут показаться не совсем понятными и очевидными, они, тем не менее, должны дать хотя бы общее представление о преимуществах использования управляющих операторов.

Итак, с чего же начать? Поскольку управляющие операторы были фактически "заимствованы" из теории программирования, хотя бы небольшой опыт написания программ может сослужить хорошую службу. Ну а тем, кто не обладает подобными навыками, я постараюсь объяснить способ использования и применения на практике четырех базовых управляющих операторов:

- оператора объявления переменной;
- оператора присвоения переменной значения;
- оператора `IF`;
- оператора `WHILE`.

Кроме этого, в состав управляющих операторов Transact-SQL входят не рассматриваемые в этой книге операторы `CASE`, `GOTO` и `WAITFOR`.

Объявление переменных

Как правило, отправной точкой изучения практически любого языка программирования является знакомство со способом объявления и использования переменных. Итак, последуем этой схеме и рассмотрим способ объявления и использования переменных в Transact-SQL.

Объявление переменной в SQL Server 2000 почти полностью аналогично объявлению переменной в подавляющем большинстве языков программирования, за исключением одного момента. В SQL Server 2000 перед именем переменной обязательно должен стоять символ (префикс) `@`.

После имени указывается тип переменной и при необходимости ее размер. В одной строке кода Transact-SQL можно разместить сразу несколько объявлений переменных (в качестве альтернативы размещению каждого объявления переменной на отдельной строке). Рассмотрим приведенные ниже примеры.

Код для запуска

```
DECLARE @MyChar CHAR(9), @MyINT INTEGER, @MyDate DATETIME
```

Размещение объявления одной переменной в отдельной строке избавляет от необходимости использования запятых, но, к сожалению, обязывает указывать в начале каждой строки ключевое слово DECLARE.

Код для запуска

```
DECLARE @MyChar CHAR(9)
DECLARE @MyINT INTEGER
DECLARE @MyDate DATETIME
```

Присвоение переменным значения с помощью оператора SET

Как правило, самым актуальным после объявления переменной является вопрос ее *инициализации*.

Термин *Инициализация* (*initialization*) — присвоение переменной ее начального значения; это один из самых распространенных терминов в большинстве языков программирования.

Одна из рекомендаций Microsoft — использовать для присвоения значения переменной оператор SET вместо оператора SELECT. Ниже рассматриваются примеры использования обоих операторов, однако, как отмечает Microsoft, оператор SET специально оптимизирован для выполнения операции присвоения. Для того чтобы присвоить значение переменной @MyChar, введите следующий код в окно ввода кода программы Query Analyzer:

Код для запуска

```
DECLARE @MyChar CHAR(9)
SET @MyChar = 'SQLServer'
```

В качестве альтернативы вместо оператора SET можно использовать оператор SELECT:

Код для запуска

```
DECLARE @MyChar CHAR(9)
SELECT @MyChar = 'SQLServer'
```

И хотя с позиций кода разница между двумя управляющими операторами совсем незначительна, при дальнейшей разработке приложения будем придерживаться рекомендаций Microsoft на этот счет; действительно, кто же еще, как не разработчик, в совершенстве знает собственный продукт?

Все это хорошо, однако остался невыясненным один из самых главных вопросов: зачем вообще может понадобиться объявление переменной и присвоение ей значения? Наиболее очевидный ответ — необходимость проверки существования некоторого значения для того, чтобы убедиться в корректности заданного условия. Каким же образом это достигается с использованием переменных? И что делать в том слу-

чае, если переменной необходимо присвоить значение, взятое из таблицы? Представьте себе ситуацию, в которой переменной необходимо присвоить значение из столбца `Firstname` таблицы `Person`, чтобы проверить, не содержит ли оно символов пробела. Подобную задачу легко решить, выполнив следующий код:

Код для запуска

```
1: DECLARE @MyChar CHAR(9)
2: SET @MyChar = (SELECT Firstname FROM Person WHERE PersonID = 1)
3: SELECT @MyChar
```

В результате выполнения такого кода SQL Server 2000 возвратит фамилию человека из таблицы `Person` с идентификационным номером 1. В строке 3 указывается необходимость вывода присвоенного переменной `@MyChar` значения. В случае отсутствия этой строки переменной `@MyChar` по-прежнему будет присваиваться значение, однако пользователь не сможет его просмотреть.

Как видите, комбинирование в коде Transact-SQL операторов управления данными и управляющих операторов не представляет никакой трудности. По мере разработки приложения SQLSpyNet рассмотрим еще немало подобных приведенному выше примеров.

Итак, что же делать после того, как переменной было присвоено значение? Ведь если "звезды зажигают, значит, это кому-нибудь нужно"!

Оператор проверки условия IF, или Так что же там хранится в переменной?

После присвоения переменной значения самым "ожидаемым" действием является сравнение этого значения с каким-либо образцом. Для этого в Transact-SQL существует специальный управляющий оператор `IF`.

Оператор `IF` является одним из наиболее распространенных во всех языках программирования. Он позволяет провести проверку истинности заданного условия и выполнить, в зависимости от исхода проверки, то или иное действие. С помощью оператора `IF` можно установить существование заданного значения, а также выяснить, равно ли оно, больше или меньше какого-либо другого значения. Оператор проверки условия `IF` состоит из двух частей: собственно оператора `IF` и оператора `ELSE`.

Оператор `ELSE` выполняется только в случае ложности заданного в операторе `IF` условия. Рассмотрим пример, в котором переменной `@MyDate` присваивается некоторое значение, а затем проводится проверка соответствия этого значения определенному условию. В зависимости от исхода проверки условия в области результата выполнения кода программы Query Analyzer будет выведено соответствующее сообщение. Итак, введите и выполните код, приведенный в листинге 4.4.

Листинг 4.4. Проверка условия и вывод результата проверки

Код для запуска

```
DECLARE @MyDate DATETIME
SET @MyDate = GETDATE()
IF @MyDate = GETDATE()
   PRINT 'It is today'
ELSE
   PRINT 'It is not today'
```

В результате выполнения кода SQL Server 2000 возвратит сообщение `It is today`, как показано на рис. 4.8.

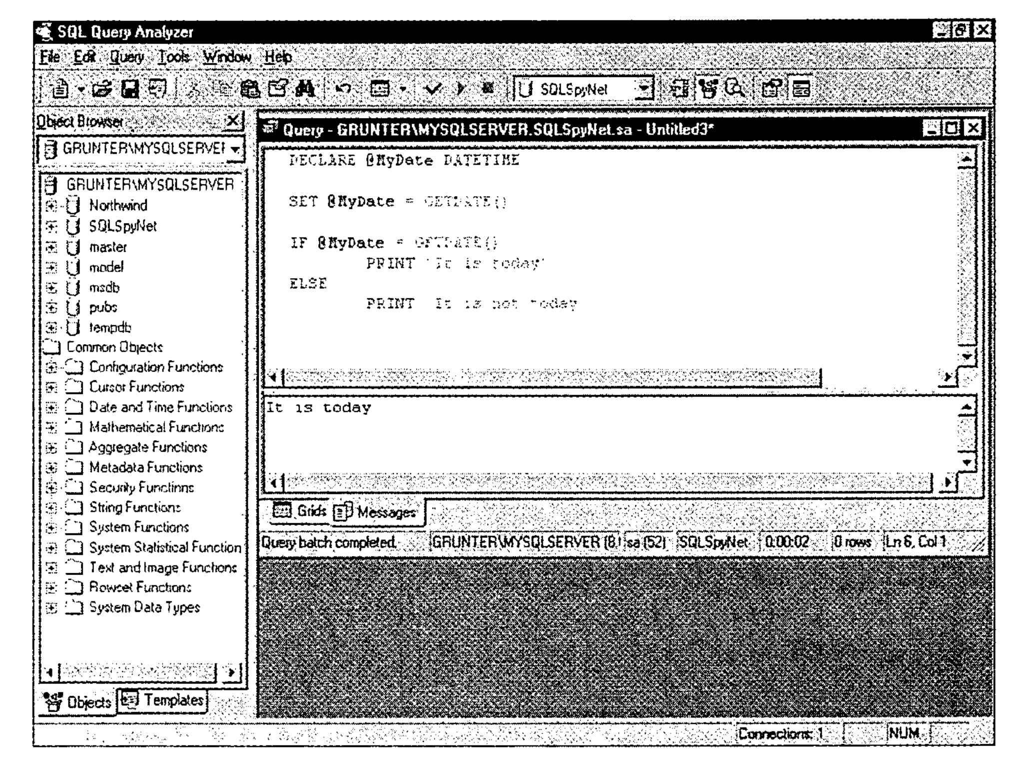


Рис. 4.8. Результат выполнения оператора `IF`

На заметку Являясь встроенной функцией SQL Server 2000, `GETDATE()` возвращает текущее значение даты на сервере. Ее аналогом может выступать функция `Now()` в языке программирования Visual Basic.

Теперь измените значение, присваиваемое переменной `@MyDate`, и вновь выполните код, обновленный вариант которого приведен в листинге 4.5.

Листинг 4.5. Проверка условия — выполнение оператора ELSE

Код для запуска

```
DECLARE @MyDate DATETIME
SET @MyDate = '01 JAN 2000'
IF @MyDate = GETDATE()
   PRINT 'It is today'
ELSE
   PRINT 'It is not today'
```

В результате выполнения кода с измененным значением переменной `@MyDate` SQL Server 2000 возвратит сообщение `It is not today`, как показано на рис. 4.9.

Выше был приведен достаточно простой, но очень наглядный пример практического использования оператора `IF...ELSE`. Уверен, что, немного поразмыслив, вы найдете массу других способов использования этого оператора. Например, оператор `IF` может быть использован для проверки существования определенной записи в таблице `BadGuy` перед обновлением информации в таблице `Spy`.

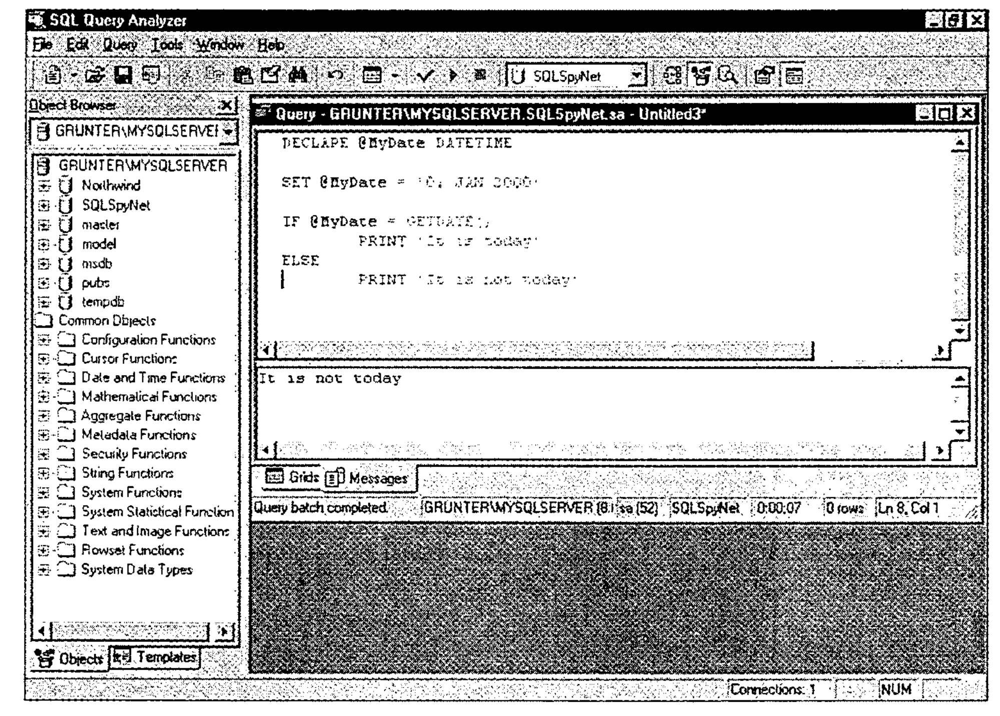


Рис. 4.9. Демонстрация выполнения оператора `ELSE`

Использование цикла WHILE

Итак, рассмотрены три из четырех основных управляющих операторов, которые предполагалось обсудить в этой части главы. Без лишнего промедления перейдем к оставшемуся управляющему оператору — `WHILE`.

Как и большинство операторов Transact-SQL, `WHILE` состоит из нескольких частей. Следует отметить, что не все части этого оператора обязательны для построения полноценного выражения `WHILE`. С позиций синтаксиса структура базового оператора `WHILE` практически аналогична структуре цикла `WHILE` в большинстве языков программирования. Цикл `WHILE` выполняется до тех пор, пока остается истинным условие цикла. Как только это условие перестает быть истинным, работа цикла прекращается.

Рассмотрим пример использования оператора `WHILE`. В результате выполнения приведенного в листинге 4.6 цикла отображаются значения счетчика.

Листинг 4.6. Использование цикла WHILE для вывода значений счетчика

Код для запуска

```
DECLARE @MyCounter INT
SET @MyCounter = 1
WHILE @MyCounter < 10
    BEGIN
       PRINT 'The Counter is now at '
         + CONVERT(CHAR(2), @MyCounter)
       SET @MyCounter = @MyCounter + 1
   END
```

SQL Server 2000 будет продолжать выполнять тело цикла до тех пор, пока условие цикла не станет ложным (рис. 4.10).

Рис. 4.10. Демонстрация выполнения оператора `WHILE`

Условие приведенного выше цикла становится в определенный момент ложным, поскольку на каждом шаге цикла происходит увеличение значения переменной `@MyCounter`.



Прежде чем запустить цикл `WHILE`, следует убедиться в том, что его условие когда-нибудь станет ложным, так как в противном случае можно попасть в довольно неприятную ситуацию, столкнувшись с так называемым *бесконечным циклом* (или, проще говоря, *зацикливанием*).

Бесконечный цикл — это цикл, никогда не прекращающий выполняться из-за того, что его условие никогда не может стать ложным. Для того чтобы прекратить выполнение бесконечного цикла, в *Query Analyzer* следует щелкнуть на расположенной на панели инструментов кнопке с изображением красного квадрата.

Анализ

Уверен, что приведенный выше код Transact-SQL уже успел заинтриговать вас своими новыми, не рассматривавшимися ранее элементами. В первую очередь это ключевые слова BEGIN...END, расположенные в строках 4 и 7, которые представляют собой еще один управляющий оператор. Функции управляющего оператора BEGIN...END полностью совпадают с функциями аналогичных конструкций во многих языках программирования, например в языке Pascal. Этот оператор позволяет сгруппировать в блок несколько операторов Transact-SQL. Ключевое слово END указывает на окончание текущего блока операторов.

Цикл WHILE требует обязательного наличия блока BEGIN...END, в котором может быть расположен один и более операторов Transact-SQL.

Следующим интересным моментом, который стоит отметить, является использование в строке 5а функции CONVERT. Эта функция позволяет изменить тип данных переменной @MyCounter с INT на CHAR длиной 2 символа. Зачем это нужно? Дело в том, что управляющий оператор PRINT может выводить на экран только данные строкового типа. Все, что не является строкой, на экран вывести нельзя.

Обратите внимание на расположенный на той же строке символ "плюс" (+). В данном случае он предназначен для формирования сообщения путем присоединения к нему значения переменной @MyCounter. Именно таким способом можно заставить SQL Server 2000 выводить каждый раз текущее значение счетчика.

Что касается метода увеличения счетчика, то я уверен, что, посмотрев на код в строке 6, вы и сами разберетесь, что к чему. Можно лишь отметить, что новое значение переменной @MyCounter формируется путем прибавления единицы к текущему значению этой переменной. Таким образом, если текущее значение переменной @MyCounter было равно 7, то ее новое значение будет получено по формуле 7+1. Все гениальное просто, не так ли?

Как упоминалось ранее, цикл WHILE состоит из нескольких частей и позволяет комбинировать другие управляющие операторы (что, кстати, и было продемонстрировано). Единственным существенным моментом, который еще не был рассмотрен, является возможность использования внутри цикла WHILE специального управляющего оператора BREAK. Этот оператор позволяет немедленно прекратить выполнение цикла в случае выполнения какого-либо заданного условия. Обратите внимание на код, приведенный в листинге 4.7, который фактически дополняет код из листинга 4.6.

Листинг 4.7. Добавление оператора BREAK в цикл WHILE

Код для запуска

```
1: DECLARE @MyCounter INT
2: SET @MyCounter = 1
3: WHILE @MyCounter < 10
4:    BEGIN
5:        PRINT 'The Counter is now at '
5a:        + CONVERT(CHAR(2), @MyCounter)
6:        SET @MyCounter = @MyCounter + 1
7:        IF @MyCounter = 7
8:            BEGIN
9:                PRINT 'Exiting the loop now'
10:                BREAK
11:            END
12:    END
```

Как показано на рис. 4.11, SQL Server 2000 продолжает выполнение цикла до тех пор, пока заданное в строке 7 условие не становится истинным.

Отметьте, что, хотя условие цикла по-прежнему остается истинным (значение переменной @MyCounter все еще меньше 10), расположенный в теле цикла оператор BREAK позволяет немедленно прекратить его выполнение.

На заметку Обратите внимание на соответствующий оператору IF блок BEGIN...END. Данный блок предназначен для группировки операторов Transact-SQL, расположенных в строках 9 и 10. Без блока BEGIN...END цикл завершит свою работу, как только достигнет оператора BREAK в строке 10. Попробуйте убрать ключевые слова BEGIN и END и убедитесь в этом сами.

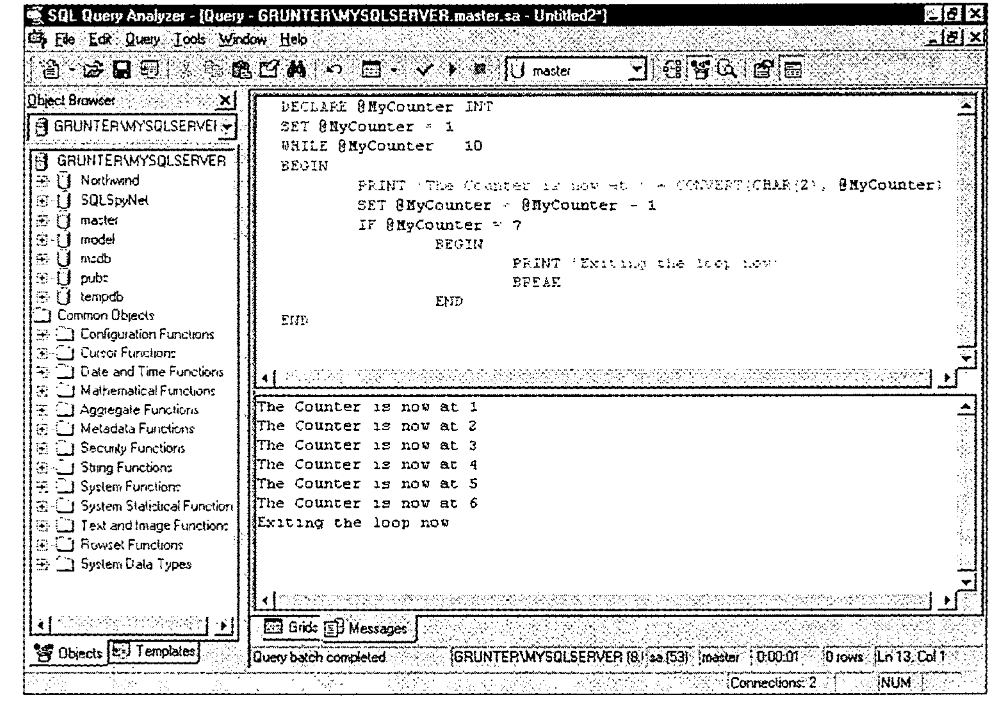


Рис. 4.11. Пример использования оператора `BREAK` *в теле цикла* `WHILE`

А зачем, собственно, может понадобиться выход из цикла `WHILE`, прежде чем его основное условие станет ложным? Дело в том, что проверка дополнительного условия внутри тела цикла не только позволяет установить истинность или ложность какого-либо значения, но и выявить некорректную или недопустимую информацию. Обнаружив ошибку, можно немедленно прервать цикл, избежав тем самым дополнительных расходов на его дальнейшее выполнение и, что еще хуже, возможных ошибок при внесении изменений в таблицу.

Итак, дамы и господа, мы только что завершили вводный курс в управляющие операторы языка Transact-SQL. Несмотря на то что "курс" вышел несколько облегченным, знание основных управляющих операторов послужит вам хорошей основой при дальнейшем изучении управляющих операторов Transact-SQL.

Резюме

В этой главе вы изучили основы языка управления данными, а также познакомились с базовыми управляющими операторами Transact-SQL. В частности, это позволило:

- выполнить оператор `SELECT` и извлечь информацию из таблицы базы данных SQLSpyNet;
- внести в базу данных несколько новых записей с помощью оператора `INSERT`;
- обновить данные таблицы с помощью оператора `UPDATE`;
- удалить из таблицы несколько строк информации с помощью оператора `DELETE`;

- объявить переменные и присвоить им значения;
- используя оператор `IF`, проверить факт существования значения и выполнить действие, зависящее от результата проверки;
- используя оператор `WHILE`, продемонстрировать способность цикла выполнять некоторую последовательность операторов Transact-SQL до наступления условия цикла;
- научиться комбинировать управляющие операторы для достижения большей гибкости при разработке приложения.

Следующие шаги

Эта глава завершает описание языка управления данными и управляющих операторов Transact-SQL. В следующей главе основное внимание уделяется языку определения данных; рассматриваются такие объекты базы данных, как представления, хранимые процедуры и триггеры. В предыдущих главах книги изучены некоторые основные операторы языка определения данных, а также такие базовые понятия, как таблица, индексы и ограничения. Объединив это с знаниями, касающимися языка управления данными и управляющих операторов Transact-SQL, постараемся разработать очень гибкие и простые в использовании операторы языка определения данных. Вы до сих пор не заинтригованы? Уверен, что все еще впереди!

Глава 5

Использование языка определения данных для просмотра и обновления информации

В этой главе...

На данный момент вы уже изучили изрядное количество операторов Transact-SQL; по моему мнению, это стало возможным благодаря четко поставленной цели — создать приложение SQLSpyNet. Естественно, было бы непростительно остановиться на достигнутом. В данной главе представлены дополнительные сведения о языке определения данных, знакомство с которым было начато еще в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet". Комбинируя операторы языка определения данных и языка управления данными, постараемся расширить функциональность и повысить гибкость разрабатываемого приложения. Первоочередные задачи, которые при этом необходимо решить, — обеспечение целостности данных и предоставление пользователям более простого способа извлечения сложных структур информации. Кроме этого, вы познакомитесь с несколькими довольно "замысловатыми" объектами базы данных.

Однако, прежде чем перейти к изучению материала этой главы, внесем в базу данных SQLSpyNet несколько новых строк информации.

Внесение новой информации в базу данных SQLSpyNet

На этот раз "пополним" таблицу `Address`. Учитывая то, что оператор внесения данных в таблицу описывался еще в предыдущей главе, приведу только пример ба-

зового оператора INSERT и предоставлю список значений, которыми должна быть заполнена таблица Address.

Итак, запустите программу Query Analyzer, выберите в качестве активной базы данных SQLSpyNet и введите код Transact-SQL, приведенный в листинге 5.1.

Листинг 5.1. Внесение информации в таблицу Address

Код для запуска

```
INSERT INTO Address (
              PersonID,
              AddressTypeID,
             CountryID,
             Address1,
             City,
             ZipCode)
VALUES (
        1,
        2,
        2,
        'Level 3, New Kelvin Chambers',
        'Wellington',
        '6001')
```

Анализ

В результате выполнения приведенного кода SQL Server 2000 возвратит сообщение, уведомляющее о внесении изменений в одну строку таблицы. Действительно, только что в таблицу Address мы внесли адрес человека из таблицы Person, идентификационный номер которого равен 1. Вскоре, как и было обещано, будет приведен полный список адресов, которыми необходимо заполнить таблицу Address базы данных SQLSpyNet.

Следует отметить, что при внесении информации в таблицу Address не стоит слишком заботиться о точном совпадении названий улиц, городов и пр., главное не ошибиться при указании значений внешних ключей таблиц, связанных с таблицей Address.

На заметку

Запомните, что самый главный момент при заполнении таблицы с внешними ключами — проверка соответствия значений этих ключей значениям первичных ключей в связанных с нею таблицах. Так, в рассматриваемом случае попытка внесения в таблицу Address записи со значением поля PersonID, не существующим в таблице Person, приведет к сообщению об ошибке и отмене внесения информации в таблицу. (Например, попробуйте внести в таблицу Address запись со значением поля PersonID, равным 3000, и посмотрите на результат.) Не правда ли, целостность на уровне ссылок — это в высшей мере полезное изобретение?



Как упоминалось ранее, вносимое в таблицу строковое значение необходимо заключить в одинарные кавычки ('). Ну а что же делать в том случае, если строковое значение само содержит символ одинарной кавычки? К примеру, как внести в таблицу строку *Rob's Address*?

Решением такой, на первый взгляд парадоксальной, ситуации будет использование вместо одной одинарной кавычки, встретившейся внутри строки, двух одинарных кавычек (''). Таким образом, в упомянутом выше случае вносимая в таблицу строка должна быть записана как *Rob''s Address*.

PersonID	AddressTypeID	CountryID	Address1	City	State	ZipCode
2	3	1	45 Somewhere Street	Los Angeles	California	
2	1	1	PO Box 123	New York		
5	1	3	PO Box 72154	London		
5	2	3	Buckingham Palace	London		
5	3	3	45 Big Ben Road	London		

Итак, заполнив новой информацией таблицу Address, можно смело приступать к изучению представлений, возвращающих пользователям базы данных SQLSpyNet как адрес, так и некоторую дополнительную информацию о наших весьма "темных" личностях.

Создание представления

Представление позволяет определить способ подачи хранящейся в базе данных информации конечному пользователю. Используя представление, можно скрыть от пользователя код Transact-SQL, необходимый для извлечения из базы данных сложных структур информации.

Во многих аспектах представление характеризуется функциями и поведением, схожими с функциями и поведением таблицы. Стоит лишь отметить, что некоторые разработчики привыкли называть представление "виртуальной таблицей". Почему же именно "виртуальной"? Дело в том, что представление само по себе не хранит никакой информации из базы данных. Все, что содержит представление, — это ссылки на таблицу (или таблицы), которая хранит нужные данные. Таким образом, при обновлении информации в представлении на самом деле обновляется информация в базовых таблицах, на которые ссылается это представление.

Благодаря схожему с таблицей поведению, представление позволяет выполнять по отношению к нему характерные для таблиц функции. Это подразумевает возможность использования таких операторов управления данными, как SELECT, INSERT, UPDATE и DELETE, хотя здесь существуют некоторые специфические ограничения.

В данном разделе рассматривается создание представления, позволяющего просмотреть адреса всех тех людей, соответствующие данные о которых занесены в таблицу Address (т.е. всех людей, которые вообще имеют адрес). Помимо адреса, представление будет извлекать из базы данных имя человека, название страны и тип адреса.

Итак, главная задача представлений — упростить способ просмотра хранящейся в базе данных информации для конечного пользователя. Представления имеют несколько существенных преимуществ:

- возможность предопределения структуры данных для просмотра конечным пользователем;
- возможность ограничения прав на просмотр представления;
- возможность использования представления для создания отчетности.

Какие же функции будет выполнять представление, разработанное специально для приложения SQLSpyNet? Представление, которое вскоре будет создано, упрощает процесс извлечения из базы данных информации об адресе конкретного человека, а также об адресах всех людей, представленных в таблице Person.

Как вы наверняка отметили еще на этапе заполнения информацией таблицы `Address`, записи об адресах существуют далеко не для каждого человека, данные о котором хранятся в таблице `Person`, вследствие чего разрабатываемое представление будет извлекать из базы данных информацию не обо всех содержащихся в ней тайных агентах и злоумышленниках.

Итак, приступим-с...

Первое знакомство с представлением

В SQL Server 2000 существует три способа создания представления: с помощью специально предназначенного для этого мастера, программы Enterprise Manager и программы Query Analyzer.

Что касается мастера, то его использование — наиболее простой подход к созданию представления. Программа Enterprise Manager, по большому счету, хороша только для новичков. Поскольку навыки, полученные в ходе освоения материала предыдущих глав, позволяют без особых трудностей работать с программой Query Analyzer, именно это средство и будем использовать для создания первого представления. Итак, если вы еще этого не сделали, запустите программу Query Analyzer и выберите в качестве активной базы данных SQLSpyNet.

Кстати, чуть не забыл! Создав представление с помощью программы Query Analyzer, вы по аналогии сможете создать представление практически в любой другой СУРБД. Таким образом, в данном случае универсальными являются не только полученные знания, но еще и сам код Transact-SQL!

При создании представления воспользуйтесь кодом, приведенным в листинге 5.2, введя его в окне ввода кода программы Query Analyzer.

Листинг 5.2. Создание представления `PersonAddress` для быстрого извлечения всей необходимой информации из базы данных

Код для запуска

```
CREATE VIEW PersonAddress AS
SELECT
     P.Firstname,
     P.Surname,
     A.Address1,
    A.City,
    A.State,
    C.Description AS Country,
    A.ZipCode,
    AType.Description AS AddressType
FROM Person P
INNER JOIN Address A
     ON P.PersonID = A.PersonID
INNER JOIN AddressType AType
    ON A.AddressTypeID = AType.AddressTypeID
INNER JOIN Country C
    ON A.CountryID = C.CountryID
```

Анализ

Как вы уже наверняка догадались, оператор `CREATE VIEW` указывает SQL Server 2000 на необходимость создания представления с именем `PersonAddress`.

Ключевое слово AS ни в коем случае не обозначает конец оператора, напротив, оно указывает на его начало. Весь следующий за этим ключевым словом код Transact-SQL полностью определяет "поведение" данного представления.

Базовый оператор SELECT уже рассматривался в главе 4, "Обработка данных с помощью кода Transact-SQL", однако в данном случае следует обратить внимание на несколько появившихся отличий (кстати, весьма интересных).

В строках 2a–2h идет перечисление столбцов, которые необходимо извлечь из базы данных. Отметьте наличие дополнительных префиксов (P., A. и т.п.) перед каждым именем столбца. Подобная практика именования столбцов известна как *использование псевдонимов*.

Термин *Использование псевдонимов* (*aliasing*) — весьма распространенный термин в большинстве языков программирования. Этот термин обозначает присвоение объекту "дружественного" имени (как правило, сокращенного). Как вы уже догадались, использование псевдонима позволяет на короткий период изменить имя объекта. SQL Server 2000 допускает назначение псевдонимов для достаточно большого количества объектов базы данных, подтверждением чему может служить назначение псевдонимов для таблиц (например, псевдоним 'A' обозначает таблицу Address) и для столбцов (например, псевдоним 'Country' соответствует столбцу C.Description) в приведенном выше определении представления PersonAddress.

Использование псевдонимов в именах столбцов позволяет явно указать таблицы, из которых были взяты эти столбцы. Следует отметить, что в этом утверждении кроется небольшая неточность, так как, например, в приведенном выше коде был использован псевдоним A, хотя в действительности таблицы с таким именем в базе данных SQLSpyNet нет.

В строках 2f и 2h определяются псевдонимы для имен столбцов. Определение псевдонима столбца позволяет назначить имя, которое этот столбец будет иметь в представлении. В данном случае столбцу Description таблицы Country был назначен псевдоним Country, а столбцу Description таблицы AddressType — псевдоним AddressType.

Однако зачем же назначать псевдоним какому-то столбцу? Дело в том, что представление, как и таблица, должно иметь уникальные имена столбцов. А поскольку таблицы Country и AddressType содержат столбцы с одинаковым названием Description, то эти столбцы не могут присутствовать в представлении со своими оригинальными именами. Таким образом, ради обеспечения уникальности имен и присваиваются данным столбцам псевдонимы, которые, кстати, достаточно открыто указывают на принадлежность столбца к той или иной таблице. С подобной практикой вы еще не раз столкнетесь при создании операторов языка определения данных.

В строке 3 определяется таблица, являющаяся источником данных для расположенного в строке 2 оператора SELECT. Обратите внимание, что после имени таблицы указывается ее псевдоним, в данном случае P. Вообще говоря, в этом примере можно было бы и отказаться от назначения псевдонима, однако его использование несколько упрощает код и делает его более наглядным. Таким образом, несколько раз встречающиеся в коде префиксы P. обозначают принадлежность соответствующего им столбца таблице Person.



Один из способов выбора псевдонимов заключается в использовании сокращенных имен объектов (в данном случае — таблиц). Как правило, это прекрасно оправдывает себя при обращении к нескольким таблицам и особенно при использовании столбцов, существующих в более чем одной таб-



лице (например, при использовании столбца PersonID, имеющегося в таблицах Person, Address, Spy и BadGuy).

Отказавшись от псевдонимов, придется указывать каждый раз полное имя таблицы, т.е., например, использовать запись Address.PersonID вместо записи A.PersonID.

Использование псевдонимов поддерживается большинством существующих на данный момент СУРБД; таким образом, знания, полученные в этом разделе, являются поистине универсальными.

Расположенный в строках 4–6а код предназначен для извлечения информации из других таблиц базы данных SQLSpyNet с помощью использования оператора INNER JOIN. Этот оператор указывает SQL Server 2000 на необходимость объединения результатов запроса с данными, которые хранятся в таблице, указанной непосредственно после ключевого слова INNER JOIN. Например, в строках 3–4а указывается необходимость объединения таблицы Person с таблицей Address. Используя оператор INNER JOIN, следует обязательно указать способ объединения двух таблиц. В случае с таблицами Person и Address объединение осуществляется по первичному ключу таблицы Person (P.PersonID) и внешнему ключу таблицы Address (A.PersonID). И это все! Таким вот нехитрым образом указываем SQL Server 2000 на необходимость объединения хранящейся в таблицах информации.

Расположенные в строках 5 и 6 операторы INNER JOIN выполняют объединение таблицы Address с двумя другими таблицами, которые содержат первичный ключ, соответствующий внешнему ключу таблицы Address.

В строках 5 и 5а выполняется объединение таблиц Address и AddressType. Это достигается путем связывания первичного ключа таблицы AddressType (AddressTypeID) и внешнего ключа таблицы Address (AddressTypeID). Аналогичным образом (т.е. используя внешний ключ одной и первичный ключ другой таблицы) в строках 6 и 6а выполняется объединение таблиц Address и Country.

Приведенный выше пример лишний раз подтверждает важность первичных и внешних ключей для реляционной базы данных. В следующем разделе рассмотрим механизм обработки SQL Server 2000 различных операторов JOIN более подробно.

Аналогично оператору SELECT, представление позволяет ограничивать возвращаемые им данные с помощью критерия WHERE, использовать который, тем не менее, пока что не будем.

В результате выполнения приведенного выше кода SQL Server 2000 создаст первое пользовательское представление базы данных SQLSpyNet. Увидеть это представление можно либо в папке Views (Представления) программы Enterprise Manager (иногда для этого может понадобиться обновить содержимое папки), либо в папке Views окна Object Browser программы Query Analyzer (содержимое этой папки также может требовать обновления).

Объединения (JOIN)

SQL Server 2000 позволяет проводить четыре базовых объединения таблиц.

- *Внутреннее объединение* (INNER JOIN) позволяет соединить две таблицы вместе и отобразить только ту информацию, которая содержится как в первой, так и во второй таблице. Это наиболее строгий из всех типов объединений, как правило используемый чаще других.

- *Левое внешнее объединение* (LEFT OUTER JOIN) позволяет объединить данные таблицы, указанной в предложении FROM, с данными таблицы, указанной в операторе JOIN. В некотором смысле левое внешнее объединение очень похоже на внутреннее объединение, разница лишь в том, что в качестве результата оператор LEFT OUTER JOIN возвращает все строки указанной в предложении FROM таблицы, отмечая значениями NULL те из них, которые не имеют соответствия в таблице, указанной в операторе JOIN. Таким образом, заменив в приведенном выше примере оператор INNER JOIN оператором LEFT OUTER JOIN, можно получить все записи из таблицы Person, в поле адреса которых будет стоять значение NULL для всех лиц, не имеющих записи об адресе в таблице Address, и соответствующий адрес для людей, имеющих запись об адресе в таблице Address. Оператор LEFT OUTER JOIN является вторым по частоте использования оператором объединения таблиц.
- *Правое внешнее объединение* (RIGHT OUTER JOIN) похоже на левое внешнее объединение, за исключением того, что в этом случае возвращаются все данные таблицы, указанной в операторе JOIN, а значениями NULL отмечается информация, не имеющая соответствия в таблице, указанной в предложение FROM.
- *Полное внешнее объединение* (FULL OUTER JOIN) подразумевает извлечение всех строк как из таблицы, указанной в предложении FROM, так и из таблицы, указанной в операторе JOIN. Результат такого объединения будет наиболее полным, несмотря на то что, с другой стороны, он будет и результатом, содержащим наибольшее количество значений NULL. Полное внешнее объединение используется наименее часто из всех рассмотренных типов объединений таблиц.

Более подробно синтаксис каждого из операторов объединения будет рассмотрен в ближайших разделах данной главы.

Поскольку объединения — материал, не слишком простой для изучения, вначале посмотрим, что можно получить с помощью использования операторов объединения таблиц.

Предположим, что есть две таблицы (например, Person и Address). Ниже приведены фрагменты таблицы Person

PersonID	FirstName	Surname	DOB	PhoneNo
1	Greg	Cross	NULL	Not Available
2	Sahara	Desert	NULL	Not Available

и таблицы Address:

PersonID	AddressTypeID	CountryID	Address1	City	State	ZipCode
1	1	1	PO Box 123	New York	NULL	NULL
1	1	3	PO Box 72154	London	NULL	NULL

Оператор INNER JOIN

В таблице Address содержится два адреса *Greg* и ни одного адреса *Sahara*. В результате выполнения приведенного ниже оператора INNER JOIN

```
SELECT P.FirstName, P.Surname, A.Address1
FROM Person P
INNER JOIN Address A ON A.PersonID = P.PersonID
```

получим следующее:

FirstName	Surname	Address1
Greg	Cross	PO Box 123
Greg	Cross	PO Box 72154

Как видите, в результат вошли только те строки таблицы Person, которые имеют соответствующие им записи (а точнее, соответствующие значения поля PersonID) в таблице Address.

Оператор LEFT OUTER JOIN

Замените в приведенном выше коде ключевое слово INNER JOIN ключевым словом LEFT OUTER JOIN:

```
SELECT P.FirstName, P.Surname, A.Address1
FROM Person P
LEFT OUTER JOIN Address A ON A.PersonID = P.PersonID
```

Выполнив код, вы получите следующий результат:

FirstName	Surname	Address1
Greg	Cross	PO Box 123
Greg	Cross	PO Box 72154
Sahara	Desert	NULL

Как видите, вновь вернулась *Sahara*, однако, поскольку соответствующей ей записи нет в таблице Address, в поле Address1 стоит значение NULL.

Оператор RIGHT OUTER JOIN

Результат выполнения оператора RIGHT OUTER JOIN во многом схож с результатом выполнения оператора LEFT OUTER JOIN, за исключением того, что значениями NULL теперь отмечается информация, не имеющая соответствия в таблице, указанной в предложении FROM. Таким образом, если бы в таблице Address содержалась информация об адресе, по которому никто не живет (т.е. в таблице Person не нашлось бы соответствующей этому адресу записи), то в объединении двух таблиц присутствовала бы строка со значениями NULL в полях Firstname и Surname и значением адреса в поле Address1.

Поскольку связь, установленная между таблицами Person и Address, подчиняется правилу целостности на уровне ссылок, то при изменении оператора объединения на RIGHT OUTER JOIN получим результат, полностью аналогичный результату, полученному в первом сценарии, т.е. при использовании оператора INNER JOIN. Чем это объяснить? Ответ более чем прост: ограничение внешнего ключа таблицы Address просто не позволяет внести в нее строку, которой бы не соответствовала запись в таблице Person. С другой стороны, также полностью исключается ситуация, в которой поле PersonID строки таблицы Address могло бы равняться NULL. Таким образом, все возвращенные в результате правого внешнего объединения таблиц Person и Address строки содержатся как в таблице Address, так и в таблице Person.

К сожалению (а если немного подумать, то окажется, что к счастью), наглядно продемонстрировать результат выполнения оператора RIGHT OUTER JOIN в сложившейся ситуации вряд ли удастся, однако следует знать, что этот оператор можно использовать для извлечения всех адресов из таблицы Address, вне зависимости от того, соответствует ли каждому из них запись в таблице Person.

Оператор FULL OUTER JOIN

Последним оператором объединения, который осталось рассмотреть, является FULL OUTER JOIN. Попробуйте догадаться, какой результат будет возвращен после выполнения приведенного ниже кода?

```
SELECT P.FirstName, P.Surname, A.Address1
FROM Person P
FULL OUTER JOIN Address A ON A.PersonID = P.PersonID
```

Не испытывайте свое терпение и посмотрите на приведенную ниже таблицу.

FirstName	Surname	Address1
Greg	Cross	PO Box 123
Greg	Cross	PO Box 72154
Sahara	Desert	NULL

Легко заметить, что результат полностью совпадает с полученным при использовании оператора LEFT OUTER JOIN. Что же случилось на этот раз? То же самое, что и при использовании предыдущего оператора объединения: аналогично тому как ограничение целостности на уровне ссылок не позволило продемонстрировать в полной мере функциональность оператора RIGHT OUTER JOIN, оно не дает воспользоваться и всеми преимуществами оператора FULL OUTER JOIN.

В самом что ни на есть классическом случае оператор FULL OUTER JOIN возвращает в результате выполнения то, что математики привыкли называть *декартовым произведением* двух множеств.

Термин *Декартово произведение* (*cartesian product*) двух множеств представляет собой набор всевозможных комбинаций элементов этих множеств. Так, если множество A состоит из элементов 1, 2 и 3, а множество B — из элементов 4, 5 и 6, то декартово произведение этих множеств будет включать в себя элементы 1, 2, 3, 4, 5 и 6.

Сравнительная характеристика представлений и таблиц

Как упоминалось ранее, представление и таблица имеют много общего. Представление позволяет извлечь хранящиеся в нем данные с помощью оператора SELECT, оно может быть объединено с таблицей и использовано в хранимой процедуре. Как и в случае с таблицей, для ограничения возвращаемой представлением информации может быть использован критерий WHERE.

Для того чтобы наглядно проиллюстрировать одно из перечисленных свойств представления, выполните приведенный в листинге 5.3 код Transact-SQL.

Листинг 5.3. Использование только что созданного представления в качестве таблицы




```
SELECT * FROM PersonAddress
```

Результат выполнения такого кода будет полностью аналогичен результату выполнения записанного в представлении оператора выборки данных SELECT. Неплохо, правда?

На заметку К сожалению, из-за использования более чем одной таблицы созданное представление не позволяет выполнять над ним другие, отличные от SELECT, операции управления данными Transact-SQL. А вот если бы использованные в представлении операторы манипулировали информацией, хранящейся только в одной таблице, то над представлением, так же, как и над таблицей, можно было бы выполнить практически любую операцию управления данными. (Более подробно ограничения представлений описываются далее в главе.)

Как вы уже наверняка догадались, только что созданное представление предназначено для просмотра пользователем адресов всех людей, данные о которых хранятся в таблице Person, не вдаваясь в подробности построения достаточно сложного запроса Transact-SQL.

Разумеется, облегчение жизни пользователя далеко не единственное преимущество при использовании представлений. Представление может предназначаться для сокрытия от конечного пользователя какой-либо важной или конфиденциальной информации. К примеру, в создаваемой нами базе данных присутствует таблица Spy с полем AnnualSalary, в котором хранится информация о заработной плате тайного агента. Совершенно понятно, что для сохранения душевного равновесия миллионов налогоплательщиков, эту информацию следует скрыть от просмотра. Сделать это можно с помощью все того же представления, которое позволяет возвратить пользователю нужную ему информацию о тайном агенте и скрыть все оставшиеся "детали". Стоит отметить, что подобного эффекта можно достичь и другими методами, которые будут описаны в самое ближайшее время.

Представления могут ссылаться внутри себя на другие представления. Кроме этого, представления могут быть созданы на основе уже существующих представлений. Все это позволяет разрабатывать невероятно сложные конструкции Transact-SQL, скрытые от глаз конечного пользователя.

Ограничения представлений

К сожалению, представления далеко не идеальны. Ниже перечислены три основных ограничения, накладываемых на представления.

- В хранящемся внутри представления операторе SELECT не разрешается использовать предложение ORDER BY.
- Представление не поддерживает передачу параметров для динамического ограничения возвращаемого представлением результата. В частности, невозможно создать представление с постоянно изменяющимся критерием WHERE.
- Обновление информации в представлении возможно только в том случае, когда представление ссылается не более чем на одну таблицу базы данных.

Несмотря на то что эти ограничения могут показаться на первый взгляд незначительными, смею вас уверить, что именно они не раз станут причиной головной боли и бессонных ночей при разработке реального приложения. Что же делать в том случае, когда какое-либо из этих ограничений станет настоящим "камнем преткновения"? Выход есть: это использование вместо представлений хранимых процедур.

Использование хранимых процедур как наиболее оптимального способа обновления информации

Хранимая процедура — это объект SQL Server 2000, представленный набором откомпилированных операторов Transact-SQL. Подобно представлению, хранимая процедура не содержит никакой информации из базы данных, вместо этого она содержит ссылки на базовые таблицы, в которых хранятся все нужные данные.

Наиболее существенное преимущество хранимой процедуры — отсутствие ограничений, присущих представлениям.

Например, хранимая процедура допускает использование предложения `ORDER BY` в хранящемся внутри нее операторе `SELECT`. Кроме этого, в хранимую процедуру можно передавать переменные для динамического изменения накладываемого на результат процедуры ограничения критерия `WHERE`.

Последнее, фактически, подразумевает возможность размещения в коде Transact-SQL ряда "вопросов" (переменных), которые будут "сняты" только перед непосредственным выполнением хранимой процедуры. Подставляя каждый раз вместо "вопросов" хранимой процедуры различные "ответы", можно выполнять одну и ту же задачу в зависимости от сложившейся ситуации, что позволяет добиться очень большой гибкости при разработке приложения.

К сожалению, хранимая процедура, подобно представлению, позволяет проводить обновление информации одновременно только в одной базовой таблице. Но, поскольку хранимая процедура — это объект, содержащий набор операторов Transact-SQL, можно обновлять каждую таблицу с помощью всего лишь одной хранимой процедуры!

Ниже перечислены преимущества хранимых процедур, которые еще не были названы в этом разделе.

- С целью защиты конфиденциальной информации пользователям дается разрешение только на выполнение хранимой процедуры, а не на доступ к содержащимся в базовых таблицах данным.
- Наряду с возможностью передачи в хранимую процедуру значений переменных, существует возможность извлечения значений из хранимой процедуры. Например, после выполнения операции по обновлению данных хранимая процедура, помимо всего прочего, может возвратить значение, сообщающее об успешном (или, наоборот, неуспешном) завершении процесса обновления информации. Теме обработки ошибок посвящена глава 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок".
- Хранимые процедуры могут содержать управляющие операторы Transact-SQL, включая объявления и присвоения значений переменным, операторы `IF`, `CASE` и `GOTO`.
- Поскольку хранимая процедура представляет собой набор операторов Transact-SQL, в одной процедуре могут одновременно присутствовать такие операторы управления данными, как `SELECT`, `INSERT`, `UPDATE` и `DELETE`.
- Поскольку все операторы хранимой процедуры предварительно компилируются еще до ее выполнения, SQL Server 2000 обладает средствами, позволяющими повысить скорость и эффективность процесса выполнения хранимой процедуры.

И хотя выше были перечислены еще не все достоинства хранимых процедур, становятся очевидными по крайней мере несколько веских причин их использования. Если говорить в общем, то с помощью хранимых процедур можно значительно повысить эффективность управления информацией в базе данных SQLSpyNet.

Экскурс

Раунд 1: хранимые процедуры против представлений

Открою вам маленький секрет: при разработке базы данных SQLSpyNet я не создал ни одного представления, реализовав все необходимые функции с помощью хранимых процедур.

Зачем я так поступил? Дело в том, что, по моему собственному (и, кстати, чрезвычайно самоуверенному) мнению, наличие представлений в SQL Server 2000, равно как и в предыдущих версиях этой СУБД, избыточно. Практически все (за исключением, правда, небольшой части), чего можно добиться с помощью представлений, может быть реализовано с использованием хранимых процедур.

Тем не менее не стоит слишком уж внимать моим словам и сбрасывать представление со всех счетов. SQL Server 2000 обладает возможностью индексирования представлений, что позволяет им выполняться более быстро и эффективно.

Трудно сказать, кто победит в этом "перетягивании каната" — представление или хранимая процедура, однако я по-прежнему остаюсь верен своему мнению и предпочитаю решать все возникающие задачи с помощью хранимых процедур.

Возможно, что этим экскурсом я уже навлек на себя гнев *многих* людей, так что пока не поздно постараюсь поподробнее объяснить свою точку зрения на сформулированную выше проблему.

В SQL Server 2000 (а также в предыдущих версиях этой СУРБД) хранимым процедурам разрешено возвращать данные. Поэтому можно включить в тело процедуры оператор выборки данных `SELECT`. Большинство других СУРБД *не поддерживают* эту возможность.

Таким образом, становится ясно, что большинство СУРБД имеют заранее запрограммированную предрасположенность к использованию представлений. В то же время множество других СУРБД не поддерживают операцию обновления базовых таблиц посредством представлений. (На этом фоне SQL Server 2000 выглядит настоящим Гулливером в стране лилипутов! Не стану скрывать — я самый что ни на есть пылкий фанат этой великолепной программы!)

Уверен, что, по большому счету, представлению попросту нечего делать в хорошо спроектированной базе данных. Несмотря на это, необходимо все-таки хотя бы уметь создавать представления для того, чтобы комплект навыков по управлению информацией в базе данных стал полноценным. К тому же многие разработчики приложений "души не чают" в представлениях, и вам наверняка не раз придется отлавливать ошибки в написанных ими программах.

Хочу еще раз подчеркнуть, что моя точка зрения ни в коем случае не является истиной в последней инстанции, которую обязательно необходимо принимать на вооружение. Многие разработчики баз данных используют для достижения поставленных целей комбинацию представлений и хранимых процедур, делая это с поразительной эффективностью и рациональностью. Таким образом, вам необходимо отсюда вынести лишь понимание того, что практически любой результат может быть достигнут с использованием различных средств.

Итак, думаю, что я привел уже достаточно объяснений своей позиции. Пора бы перейти к делу. Какие функции будут выполнять хранимые процедуры (их, как вы уже догадались, будет несколько, а именно 2), которые мы собираемся создать в следующем разделе? Обе хранимые процедуры будут выполнять похожие функции: принимать переданные значения переменных и вносить их в базовые таблицы базы данных SQLSpyNet.

Поскольку на текущий момент мы заполнили информацией таблицу Person, оставив без внимания Spy и BadGuy, именно они и будут рассматриваться в качестве таблиц для внесения информации. Первая хранимая процедура будет вносить данные в таблицу Person и одновременно в таблицу Spy, заполняя, таким образом, сразу две таблицы! Вторая хранимая процедура будет выполнять аналогичные действия по отношению к таблицам Person и BadGuy.

Создание кода хранимой процедуры

Создадим хранимую процедуру, которая обновляет информацию в таблицах Person и Spy. Что касается синтаксиса Transact-SQL, то обе хранимые процедуры, рассматриваемые в данном разделе, достаточно просты, так как практически все используемые в них операторы представлены уже изученными ранее операторами управления данными.

На заметку

Следует помнить, что хранимые процедуры, равно как и представления, таблицы и индексы, являются объектами базы данных. Таким образом, при использовании хранимой процедуры сначала необходимо создать соответствующий объект, а затем выполнить его (используя операторы управления данными Transact-SQL), указав при этом все необходимые значения параметров.

Итак, запустите программу Query Analyzer, выберите SQLSpyNet в качестве активной базы данных и введите код из листинга 5.4.

Листинг 5.4. Создание кода заголовка хранимой процедуры

Код для запуска

```
1: CREATE PROCEDURE PersonSpyInsert
2:     @Firstname VARCHAR(50),
3:     @Surname VARCHAR (50),
4:     @DOB DATETIME = NULL,
5:     @SpyNumber VARCHAR(10) = NULL,
6:     @Alias VARCHAR(25) = NULL,
7:     @DateCommencedWork DATETIME,
8:     @AnnualSalary MONEY,
9:     @IsActive BIT = 1
10: AS
```

Анализ

Строка 1 приведенного выше кода указывает SQL Server 2000 на необходимость создания хранимой процедуры с именем PersonSpyInsert. При выборе имени хранимой процедуры я придерживался правила именования объектов, в соответствии с которым имя состоит из имен объектов, изменяемых хранимой процедурой, и из названия выполняемого хранимой процедурой действия.

Следующая после объявления имени хранимой процедуры часть кода предназначена для указания переменных (называемых также *параметрами*), значения которых будут переданы в хранимую процедуру при ее выполнении.

В строках 2–9 размещены объявления переменных, включающие определения типов данных и значений по умолчанию. Подобным образом мы объявляли и инициализировали переменные в главе 4, "Обработка данных с помощью кода Transact-SQL", единственное отличие состоит в том, что в данном случае не нужно использовать ключевое слово DECLARE.

Обратите внимание: объявленная в строке 4 переменная @DOB имеет по умолчанию значение NULL. Это означает, что если при вызове хранимой процедуры значение этой переменной будет опущено, то по умолчанию она будет инициализирована значением NULL. Подобное объявление переменной присутствует также и в строке 9. Строка 10 приведенного выше кода аналогична соответствующей строке в рассмотренном ранее операторе CREATE VIEW. Весь последующий за ключевым словом AS код Transact-SQL полностью определяет тело, а значит, и функциональность хранимой процедуры.

Что же дальше? После определения заголовка хранимой процедуры следует ввести код Transact-SQL, необходимый для проведения обновлений данных в таблицах Person и Spy. Итак, введите приведенный в листинге 5.5 код в окно ввода кода программы Query Analyzer.

Листинг 5.5. Создание хранимой процедуры PersonSpyInsert для одновременного внесения информации в таблицы Person и Spy

Код для запуска

```
1:  CREATE PROCEDURE PersonSpyInsert
2:     @Firstname VARCHAR(50),
3:     @Surname VARCHAR (50),
4:     @DOB DATETIME = NULL,
5:     @SpyNumber VARCHAR(10) = NULL,
6:     @Alias VARCHAR(25) = NULL,
7:     @DateCommencedWork DATETIME,
8:     @AnnualSalary MONEY,
9:     @IsActive BIT = 1
10:  AS
11:  DECLARE @PersonID INT
12:  INSERT INTO Person (Firstname, Surname, DOB)
13:  VALUES (@Firstname, @Surname, @DOB)
14:  SET @PersonID = IDENT_CURRENT('Person')
15:  INSERT INTO Spy (PersonID, SpyNumber, Alias,
15a:  DateCommencedWork, AnnualSalary, IsActive)
16:  VALUES (@PersonID, @SpyNumber, @Alias,
16a:  @DateCommencedWork, @AnnualSalary, @IsActive)
```

Анализ

Наименьшие трудности при рассмотрении приведенного выше кода наверняка вызовет оператор INSERT, который был досконально изучен в главе 4, "Обработка данных с помощью кода Transact-SQL". Единственным отличием от классического варианта использования этого оператора является наличие переменных в предложении VALUES, что позволяет изменять вносимую в таблицы Person и Spy информацию, не переписывая всего оператора Transact-SQL. Поскольку в заголовке хранимой процедуры были определены типы всех переменных, можно не заботиться о заключении в кавычки вносимых в таблицы строковых значений.

Рассмотрим способ внесения информации в таблицу Spy (поскольку с таблицей Person вроде бы все ясно).

Наиболее важный момент, касающийся внесения информации в таблицу Spy, — необходимость извлечения из таблицы Person значения соответствующего только что внесенным данным первичного ключа PersonID.

Обратите внимание на объявление переменной @PersonID в строке 11. В строке 14 расположен оператор, присваивающий значение этой переменной, равное значению столбца PersonID в только что внесенной в таблицу Person новой записи. Поскольку PersonID является идентификационным столбцом таблицы Person, следует воспользоваться некоторым специальным методом извлечения информации из этого столбца.

SQL Server 2000 предоставляет несколько различных методов извлечения информации из идентификационного столбца таблицы, однако в данном случае для выполнения этой операции воспользуемся функцией IDENT_CURRENT (более подробно о функциях SQL Server 2000 речь пойдет в главе 6, "Использование функций для повышения эффективности управления информацией"). В качестве аргумента эта функция принимает имя таблицы, а в качестве результата возвращает значение первичного ключа последней внесенной в таблицу записи.

Значение первичного ключа PersonID последней внесенной в таблицу Person записи необходимо для создания соответствующего ему внешнего ключа таблицы Spy, что позволит установить связь между записями обеих таблиц. Наличие такой связи крайне важно для целостности базы данных вообще и для проведения объединений таблицы Spy с другими таблицами в частности.

Значение первичного ключа таблицы Person присваивается локальной переменной @PersonID и передается в качестве параметра в следующий оператор INSERT. Напомню, что встроенные функции SQL Server 2000 обсуждаются в следующей главе, так что рассмотрение некоторых интересных моментов приведенного выше кода придется пока отложить.

В сроках 15–16а расположен оператор INSERT, осуществляющий внесение информации в таблицу Spy. Выполните код, в результате чего SQL Server 2000 возвратит сообщение, подтверждающее успешное создание хранимой процедуры, как показано на рис. 5.1.

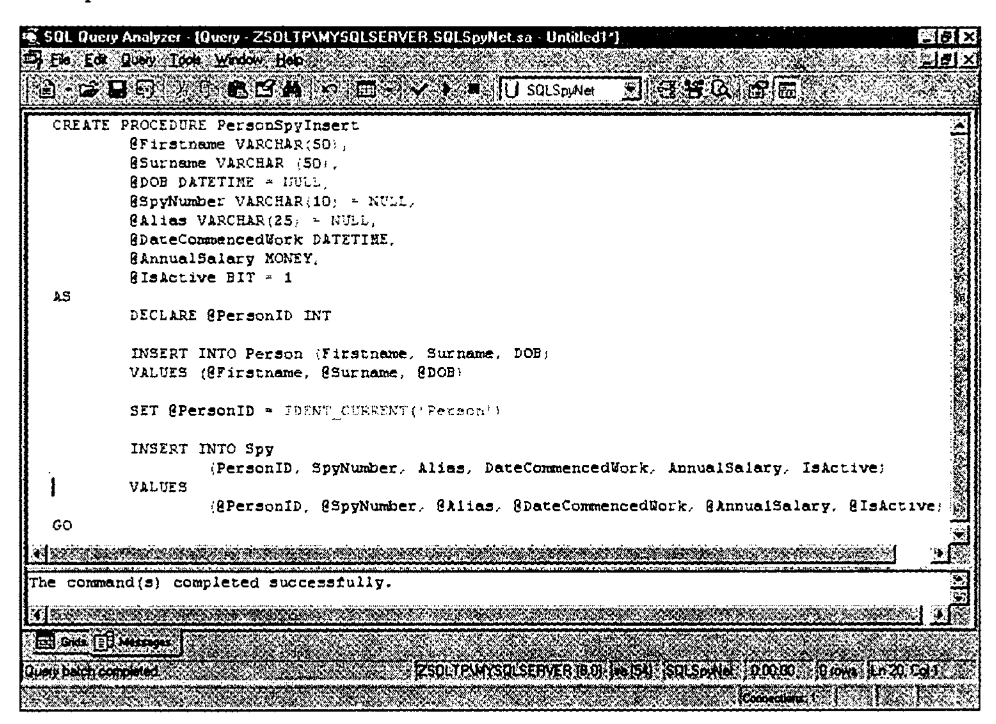


Рис. 5.1. SQL Server 2000 подтверждает факт создания хранимой процедуры

Проверка возможности внесения информации в таблицы Person и Spy

Для того чтобы удостовериться в работоспособности только что созданной хранимой процедуры, ее необходимо хотя бы один раз выполнить. Но перед этим давайте осуществим дополнительную проверку хранящейся в таблицах Person и Spy информации и убедимся в том, что ее внесение, по крайней мере теоретически, пройдет корректно. Введите и выполните код Transact-SQL, приведенный в листинге 5.6.

Листинг 5.6. Проверка хранящейся в таблицах Person и Spy информации, цель которой — убедиться в теоретической работоспособности хранимой процедуры PersonSpyInsert

Код для запуска

```
SELECT P.PersonID, S.SpyID, P.Firstname FROM Person P
LEFT OUTER JOIN Spy S ON S.PersonID = P.PersonID
```

Результат выполнения кода показан на рис. 5.2.

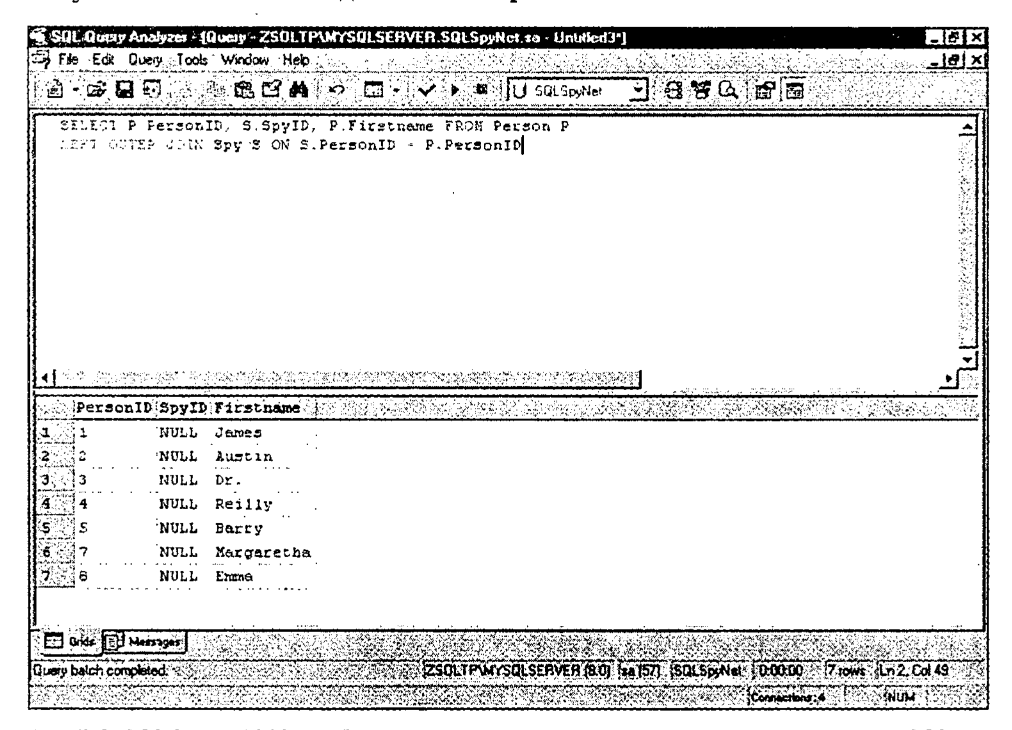


Рис. 5.2. SQL Server 2000 отображает результат выполнения оператора Transact-SQL

Обратили внимание на большое количество значений NULL в столбце SpyID? Это свидетельствует об отсутствии информации о тайных агентах в таблице Spy.

Использование в приведенном выше коде оператора LEFT OUTER JOIN оправдывается необходимостью извлечения всей информации из таблицы Person при условии, что таблица Spy не содержит никаких данных. Если бы вместо оператора LEFT

OUTER JOIN использовался INNER JOIN, SQL Server 2000 не смог бы найти ни одной строки таблицы Person, имеющей соответствующую ей запись в таблице Spy.

Выполнение хранимой процедуры

Для того чтобы выполнить хранимую процедуру PersonSpyInsert, введите код из листинга 5.7 в окно ввода кода программы Query Analyzer.

Листинг 5.7. Внесение информации в таблицы Person и Spy с помощью запуска хранимой процедуры PersonSpyInsert

Код для запуска

```
EXEC PersonSpyInsert 'Dangerous',
'Mouse',NULL,NULL,NULL,'15 Jul 2000',75000.00
```

Анализ

Обратите внимание на то, что в хранимую процедуру передаются только те параметры, для которых не было определено значение по умолчанию. Несмотря на то что при создании хранимой процедуры задано достаточно большое количество параметров, при вызове процедуры указывать их все не обязательно.

Еще раз выполнив приведенный в листинге 5.6 код, вы увидите новую строку со значением столбца Firstname, равным *Dangerous*, и значением столбца SpyID (рис. 5.3).

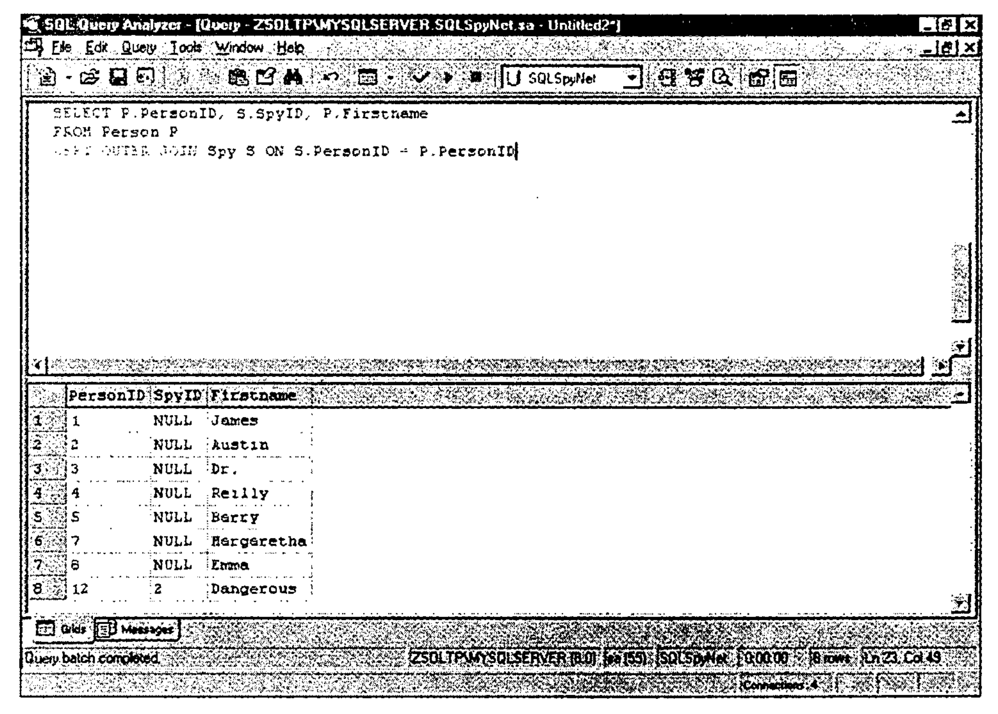


Рис. 5.3. SQL Server 2000 отображает результат выполнения приведенного в листинге 5.6 оператора Transact-SQL после внесения информации в таблицы Person *и* Spy *с помощью хранимой процедуры* PersonSpyInsert

Здорово, не правда ли? Информация о тайном агенте была внесена одновременно в две таблицы, причем для этого потребовалась всего лишь одна строка кода Transact-SQL. И хотя приведенный выше рисунок еще не дает стопроцентного доказательства того, что обновление информации состоялось как в таблице Person, так и в таблице Spy, можете не сомневаться, что это так и есть. В крайнем случае выполните с помощью оператора SELECT выборку данных из таблиц Person и Spy, чтобы воочию убедиться в полной работоспособности хранимой процедуры PersonSpyInsert.

Создание хранимой процедуры для одновременного внесения информации в таблицы Person и BadGuy

Хранимая процедура PersonBadGuyInsert во многом аналогична рассмотренной ранее хранимой процедуре PersonSpyInsert. Введите код из листинга 5.8 в окно ввода кода программы Query Analyzer.

Листинг 5.8. Создание хранимой процедуры PersonBadGuyInsert для одновременного внесения информации в таблицы Person и BadGuy

Код для запуска

```
CREATE PROCEDURE PersonBadGuyInsert
   @Firstname VARCHAR(50),
   @Surname VARCHAR (50),
   @DOB DATETIME = NULL,
   @KnownAs VARCHAR(25) = NULL,
   @IsActive BIT = 1
AS
   DECLARE @PersonID INT
   INSERT INTO Person (Firstname, Surname, DOB)
    VALUES (@Firstname, @Surname, @DOB)
    SET @PersonID = IDENT_CURRENT('Person')
    INSERT INTO BadGuy (PersonID, KnownAs, IsActive)
    VALUES (@PersonID, @KnownAs, @IsActive)
```

Анализ

Как уже отмечалось, хранимая процедура PersonBadGuyInsert во многом аналогична хранимой процедуре PersonSpyInsert, что существенно упрощает разбор структуры и анализ приведенного выше кода.

Что же отличает PersonBadGuyInsert от рассмотренной ранее хранимой процедуры PersonSpyInsert? Единственное и самое главное отличие заключается в том, что на этот раз обновление информации выполняется в таблицах BadGuy и Person. Соответственно несколько уменьшился список параметров, передаваемых в хранимую процедуру PersonBadGuyInsert, по сравнению со списком параметров, передаваемых в хранимую процедуру PersonSpyInsert.

Результат выполнения кода, приведенного в листинге 5.8, будет таким, как на рис. 5.1.

Проверка возможности внесения информации в таблицы Person и BadGuy

Проведем дополнительную проверку информации, хранящейся в таблицах Person и BadGuy, чтобы удостовериться в теоретически корректном выполнении хранимой процедуры PersonBadGuyInsert. Введите и выполните код, приведенный в листинге 5.9.

Листинг 5.9. Проверка хранящейся в таблицах Person и BadGuy информации, подтверждающая теоретически корректное выполнение хранимой процедуры PersonBadGuyInsert

Код для запуска

```
SELECT P.PersonID, B.BadGuyID, P.Firstname
FROM Person P
LEFT OUTER JOIN BadGuy B ON B.PersonID = P.PersonID
```

Результат выполнения кода показан на рис. 5.4.

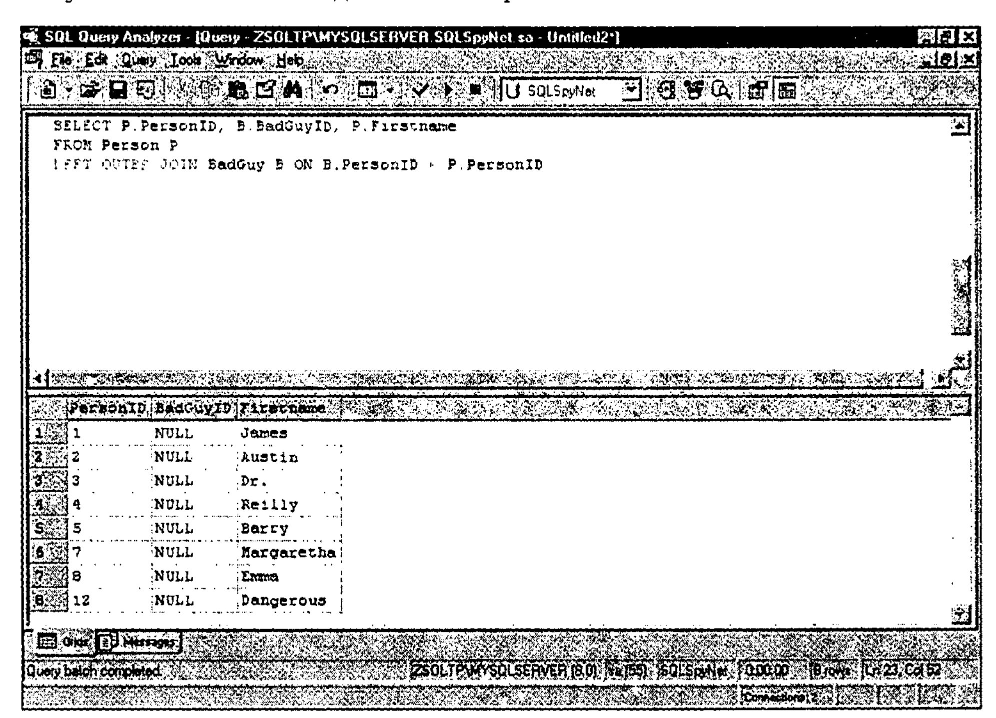


Рис. 5.4. SQL Server 2000 отображает результат выполнения оператора Transact-SQL перед выполнением хранимой процедуры PersonBadGuyInsert

Обратите внимание на большое количество значений NULL (аналогично тому, как это было после выполнения кода, приведенного в листинге 5.6) в возвращенном SQL Server 2000 результате объединения таблиц Person и BadGuy. Значения NULL свидетельствуют об отсутствии записей о злоумышленниках в таблице BadGuy.

Для того чтобы выполнить хранимую процедуру PersonBadGuyInsert (а заодно избавиться от некоторых значений NULL), введите и выполните код Transact-SQL, приведенный в листинге 5.10.

Листинг 5.10. Внесение информации в таблицы Person и BadGuy посредством выполнения хранимой процедуры PersonBadGuyInsert

Код для запуска ⟶

```
1: EXEC PersonBadGuyInsert 'Greg',
1a:'Cross', NULL, 'Gregster', 1
```

Выполните еще раз оператор SELECT, приведенный в листинге 5.9; полученный результат изображен на рис. 5.5.

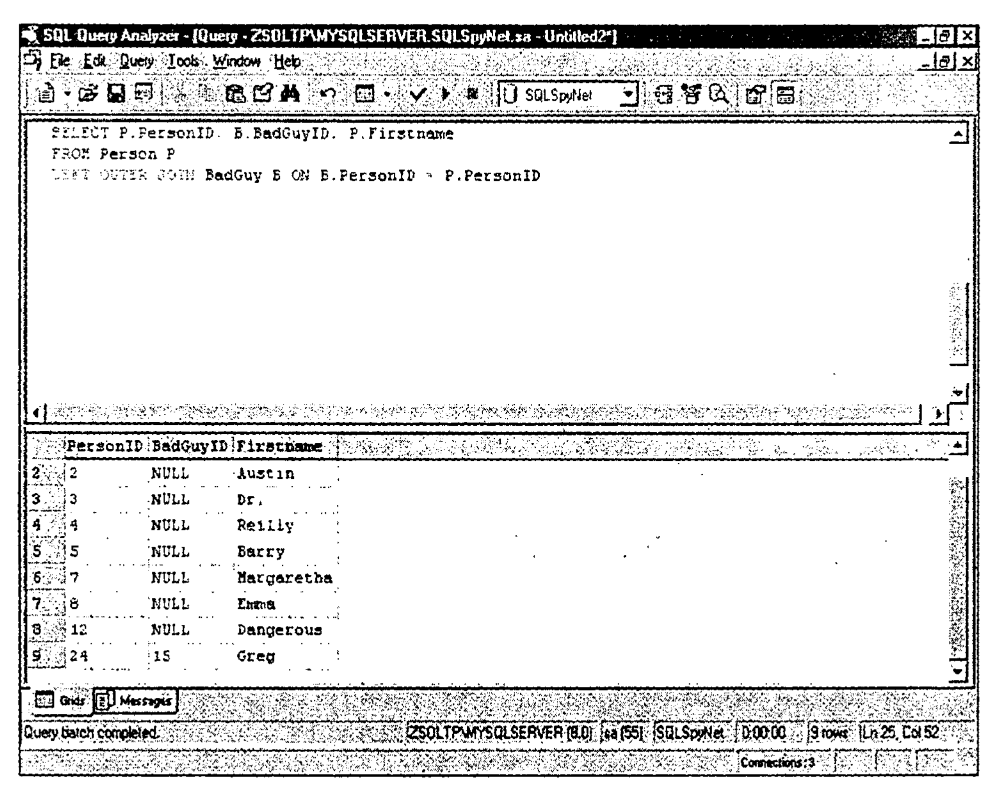


Рис. 5.5. SQL Server 2000 отображает результат выполнения оператора Transact-SQL после внесения новой информации в таблицы Person *и* BadGuy *с помощью хранимой процедуры* PersonBadGuyInsert

Как и в случае выполнения хранимой процедуры PersonSpyInsert, в результате выполнения PersonBadGuyInsert новая информация была занесена одновременно в две таблицы: на этот раз в таблицы Person и BadGuy.

Итак, дамы и господа, вы только что создали две хранимые процедуры и убедились в их работоспособности. Ай да молодцы!

Некоторые соображения по поводу использования хранимых процедур

Несмотря на то что разработка базы данных с использованием только хранимых процедур — задача, технически вполне осуществимая, было бы слишком опрометчиво намеренно лишать себя всех оставшихся возможностей SQL Server 2000.

При всей своей эффективности и универсальности использования хранимая процедура лишена одного очень важного свойства: способности реагировать на события. Это, в частности, обозначает невозможность автоматического выполнения хранимой процедуры при внесении в таблицу новой строки информации. Способностью реагировать на события обладают специальные объекты базы данных SQL Server 2000, называемые *триггерами*, обсуждению которых посвящен весь следующий раздел этой главы.

Дальнейшая разработка приложения SQLSpyNet требует создания еще нескольких хранимых процедур, код Transact-SQL которых будет приведен без каких-либо детальных объяснений. Исключения будут сделаны только для хранимых процедур, содержащих реализацию некоторых новых, не рассматривавшихся ранее идей.

Как вы уже, вероятно, догадались, хранимые процедуры будут использоваться в основном для обновления и извлечения информации из базы данных. С помощью таких "передовых" технологий компании Microsoft, как ADO (ActiveX Data Objects), вызывать хранимые процедуры можно из пользовательских приложений. Естественно, что обработка и передача данных объекту ADO намного эффективнее раздельного создания и передачи данных операторам Transact-SQL. Более подробно объекты ADO рассматриваются в следующих главах в рамках создания пользовательского интерфейса приложения SQLSpyNet.

Итак, тема использования хранимых процедур в SQL Server 2000 исчерпана. Как я и обещал, следующий раздел этой главы целиком посвящается триггерам.

Обработка событий базы данных с помощью триггеров

Так что же такое триггер? (Надо признаться, это слово вызывает ассоциации с каким-нибудь хитроумным устройством из арсенала Джеймса Бонда.)

По большому счету, триггер можно рассматривать как особый тип хранимой процедуры, способной реагировать на *события*.

Термин *События* (*events*) возникают вследствие выполнения некоторых действий. (Если вы разрабатывали программы для операционной системы Windows или использовали какой-либо из распространенных языков сценариев, то концепция событий должны быть вам хорошо знакома.) Например, при открытии окна возникает связанное с этим действием событие `onOpen`. Аналогичным образом, при щелчке на кнопке возникает событие `onClick`. Безусловно, все это несколько утрировано, однако в большинстве своем приведенные выше примеры отражают реальное положение вещей.

В отличие от Windows, в SQL Server 2000 событие возникает в результате выполнения некоторого действия, связанного с таблицей. Например, событием является

внесение в таблицу новой строки информации. Событием также является и удаление строки из таблицы.

При чем же здесь триггеры? Как вы помните, триггеры обладают возможностью реагировать на события. Это означает, например, что можно запустить триггер при внесении информации в таблицу для того, чтобы проверить достоверность этой информации или определить наличие связанных с ней данных в других таблицах. При удалении данных из таблицы можно было бы использовать триггер, удаляющий все связанные с этими данными записи из других таблиц базы данных.

На заметку Благодаря новым возможностям, появившимся в SQL Server 2000, удаление всей связанной с определенной записью информации происходит автоматически, не требуя написания для этого специального триггера.

Напомню, что при установке отношений между таблицами с помощью специального средства построения диаграмм базы данных мы обращали внимание на параметр, позволяющий SQL Server 2000 автоматически поддерживать целостность базы данных на уровне ссылок, удаляя все дочерние записи в базе данных при удалении из нее соответствующей им родительской записи. Тем не менее до сих пор мы не использовали всех преимуществ этой замечательной возможности.

Одно из самых существенных нововведений SQL Server 2000 — возможность назначать несколько различных триггеров одному и тому же событию таблицы (следует отметить, что большинство других СУРБД пока что не могут этим похвастаться).

Традиционный подход при использовании триггеров заключается в назначении одного триггера одному событию таблицы. Это (к счастью для пользователей SQL Server 2000, уже в прошлом) подразумевает необходимость написания очень сложных триггеров для реализации большинства возникающих при разработке реальных баз данных задач. Упомянутое выше свойство SQL Server 2000 позволяет сделать триггеры более простыми и компактными. Помимо этого, в SQL Server 2000 реализован новый тип триггеров, более подробно описанный в главе 15, "Самостоятельное исследование SQL Server 2000".

Создание триггера, "разоблачающего" двойных агентов

Триггеры, рассматриваемые в этом разделе, будут ассоциированы с таблицами BadGuy и Spy. Следует признать, что на текущий момент все еще остается нерешенным один довольно щекотливый вопрос. Суть проблемы заключается в модели данных приложения SQLSpyNet, предусматривающей определение принадлежности человека к тайным агентам (таблица Spy) или к злоумышленникам (таблица BadGuy). Если оставить все как есть, то можно запросто попасть в ситуацию, когда один и тот же человек будет одновременно отнесен и к тайным агентам, и к злоумышленникам. Несмотря на то что в реальном мире так называемые двойные агенты встречаются не так уж редко, в разрабатываемом приложении подобная возможность должна быть полностью исключена. (Действительно, покажите мне хотя бы одного директора организации по найму тайных агентов, который возьмет к себе на работу двойника!)

Чтобы предупредить такую ситуацию, создадим триггер для таблицы BadGuy (а также для таблицы Spy), который будет проверять наличие злоумышленника с заданным именем в таблице Spy (и наоборот, тайного агента с заданным именем —

в таблице BadGuy). В случае положительного исхода проверки триггер будет возвращать сообщение об ошибке с указанием причины отказа в проведении операции над таблицей. Следует отметить, что для полной уверенности данный триггер будет выполняться каждый раз не только при внесении информации в таблицу (событие INSERT), но также и при ее изменении (событие UPDATE).

На заметку При создании триггера для таблицы необходимо определить событие, с которым будет ассоциирован этот триггер. Другими словами, должен ли триггер выполняться в ответ на попытку внесения (INSERT), изменения (UPDATE) или удаления (DELETE) информации из таблицы?

Рассматриваемые в этом разделе триггеры относятся к триггерам типа AFTER, которые выполняются после внесения изменений в таблицу. При обнаружении ошибки изменения могут быть отменены с помощью операции так называемого отката (см. главу 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок").

К новым особенностям SQL Server 2000 относится поддержка триггеров, выполняющихся перед внесением изменений в данные таблицы (более подробно этот тип триггеров обсуждается в главе 15, "Самостоятельное исследование SQL Server 2000"). Триггеры типа INSTED OF (а именно так они называются) позволяют "на лету" проанализировать корректность вносимых в таблицу изменений и затем, в зависимости от результата проверки, осуществить эти изменения удобным способом.

Итак, перейдем непосредственно к самому процессу создания триггеров. Выберите SQLSpyNet в качестве активной базы данных и введите код из листинга 5.11 в окно ввода кода программы Query Analyzer.

Листинг 5.11. Синтаксис оператора CREATE TRIGGER первого триггера базы данных SQLSpyNet

Код для запуска

```
CREATE TRIGGER CheckBadguyNotInSpy
ON BadGuy
FOR INSERT, UPDATE
AS
   BEGIN TRANSACTION
   DECLARE @SpyID INT
   DECLARE @PersonID INT
   SELECT @PersonID = i.PersonID
   FROM inserted i
    SELECT @SpyID = SpyID
    FROM Spy S
    WHERE S.PersonID = @PersonID
    IF (@SpyID IS NOT NULL) AND (@SpyID > 0)
        BEGIN
            RAISERROR ('The Person you are trying to
Insert/Update already exists in the Spy table.',
16, 1)
            ROLLBACK TRANSACTION
        END
    ELSE
        BEGIN
            COMMIT TRANSACTION
        END
```

Анализ Поскольку на текущий момент у вас уже есть общее представление о создании объектов базы данных, нет смысла останавливаться на строке 1.

А вот строка 2 уже требует того, чтобы ей уделили немного внимания. При создании триггер ассоциируется с некоторым событием. Поскольку событие возникает вследствие совершения действия над таблицей, при определении триггера необходимо указать имя таблицы, для которой он предназначается. В данном случае в строке 2 указывается, что триггер CheckBadguyNotInSpy предназначен для таблицы BadGuy.

В строке 3 определяются события, с которыми должен быть ассоциирован триггер. Обратите внимание на то, что триггер CheckBadguyNotInSpy ассоциируется сразу с двумя событиями таблицы BadGuy — INSERT и UPDATE. При желании можно связать триггер со всеми событиями таблицы (DELETE, INSERT и UPDATE) или же только с одним из них.

Назначение расположенного в строке 4 ключевого слова AS полностью совпадает с его назначением в объявлениях представлений и хранимых процедур.

В строке 5 расположен специальный оператор, который более подробно рассматривается в одной из следующих глав книги. Главная функция данного оператора — гарантировать возможность возврата к предыдущему состоянию таблицы в случае внесения (INSERT) в нее некорректной информации.

На заметку Более подробно транзакции рассматриваются в главе 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок".

В строках 6 и 7 располагаются объявления переменных, которые будут использованы для хранения значения столбца PersonID только что внесенной в таблицу записи и значения столбца SpyID таблицы Spy.

Почти всю оставшуюся часть триггера занимает оператор SELECT, предназначенный для проверки наличия записи с заданным идентификационным номером в таблице Spy. Несмотря на то что данный оператор был досконально изучен еще в предыдущей главе, обратите особое внимание на строки 8 и 9.

Расположенный в этих строках оператор SELECT выполняет функцию присвоения переменной @PersonID значения столбца PersonID только что внесенной в таблицу BadGuy записи. Несложная с первого взгляда операция выполнена весьма интересным способом. Дело в том, что при выполнении операций вставки (INSERT), изменения (UPDATE) или удаления (DELETE) информации из базы данных SQL Server 2000 создает в памяти две таблицы: inserted и deleted.

Таблица deleted используется при удалении строки из базовой таблицы. Она содержит копию удаленной строки, позволяя таким образом обратиться (именно обратиться, так как таблицы inserted и deleted не позволяют изменять хранящуюся в них информацию) к только что удаленным данным.

Аналогичным образом в таблице inserted хранится копия строки, только что внесенной в базовую таблицу. Именно этот факт и используется в строке 9 при извлечении значения столбца PersonID только что внесенной в таблицу BadGuy записи.

На заметку При выполнении операторов INSERT и DELETE SQL Server 2000 использует таблицы inserted и deleted отдельно друг от друга. Однако при выполнении оператора UPDATE эти таблицы используются вместе. Чем это объяснить? Дело в том, что операция обновления данных состоит из двух фаз: сначала текущая информация удаляется (при этом ее копия помещается в таблицу deleted), после чего на ее место записываются новые данные (копия которых содержится в таблице inserted). Таким образом, обновляя

На заметку

информацию в таблице с помощью оператора UPDATE, следует воспользоваться преимуществами обеих временных таблиц.

При использовании таблиц inserted и deleted необходимо иметь в виду один очень важный момент: они не являются постоянными таблицами SQL Server 2000 и существуют в памяти очень небольшой промежуток времени — ровно столько, сколько необходимо для выполнения группы операторов Transact-SQL. Кроме этого, при написании имен таблиц inserted и deleted следует использовать только строчные символы.

В строке 13 расположен управляющий оператор `IF`, который осуществляет проверку значения переменной `@SpyID`. Если переменная не имеет никакого значения, то выполнение триггера переходит в строку 18. В строке 20 расположен оператор, подтверждающий правомочность изменения информации в таблице.

Код, расположенный в строках 15–16, отвечает за уведомление пользователя о случившейся ошибке. Для этого используется встроенная функция SQL Server 2000 `RAISERROR`. Более подробно эта функция обсуждается в главе 7, "Защита пользовательского ввода от возможных ошибок с помощью стандартных значений и правил". И наконец, расположенный в строке 16 оператор отменяет операцию вставки/изменения информации в таблице, осуществляя возврат к ее старым значениям.

Вот и все, что касается создания первого триггера базы данных SQLSpyNet. Его главное назначение — поддержка целостности хранящейся в базе данных информации, а значит, и работоспособности всего создаваемого приложения.

Тестирование триггера

Для того чтобы убедиться в работоспособности только что созданного триггера, его необходимо протестировать. Однако перед этим внесем в таблицы `Spy` и `BadGuy` по одной новой записи. Затем попробуем добавить в таблицу `BadGuy` двойного агента и посмотрим, сумеет ли триггер "отстоять" целостность хранящейся в базе данных SQLSpyNet информации.

Введите код листинга 5.12 в окно ввода кода программы Query Analyzer.

Листинг 5.12. Внесение новой записи в таблицу `BadGuy` для проверки работоспособности триггера `CheckBadguyNotInSpy`

Код для запуска

```
INSERT INTO BadGuy (PersonID, KnownAs, IsActive)
VALUES (3, NULL, 1)
```

Легко видеть, что приведенный выше код вносит новую запись в таблицу `BadGuy` (эта запись понадобится при проверке работоспособности триггера `CheckBadguyNotInSpy`).

Теперь введите код из листинга 5.13.

Листинг 5.13. Внесение новой записи в таблицу `Spy` для проверки работоспособности триггера `CheckBadguyNotInSpy`

Код для запуска

```
INSERT INTO Spy (PersonID, AnnualSalary, IsActive)
VALUES (1, 80000.00, 1)
```

Приведенный выше код вносит новую запись в таблицу Spy (эта запись также понадобится при проверке работоспособности триггера CheckBadguyNotInSpy). А теперь, поскольку мы уже подготовили все необходимые для этого данные, выполните код листинга 5.14, который предназначен непосредственно для проверки работоспособности созданного триггера.

Листинг 5.14. Попытка внесения информации о двойном агенте в таблицу BadGuy

Код для запуска

```
INSERT INTO BadGuy (PersonID, KnownAs, IsActive)
VALUES (1, NULL, 1)
```

На этот раз выполнение приведенного кода завершится тем, что SQL Server 2000 возвратит определенное в триггере CheckBadguyNotInSpy сообщение об ошибке (рис. 5.6).

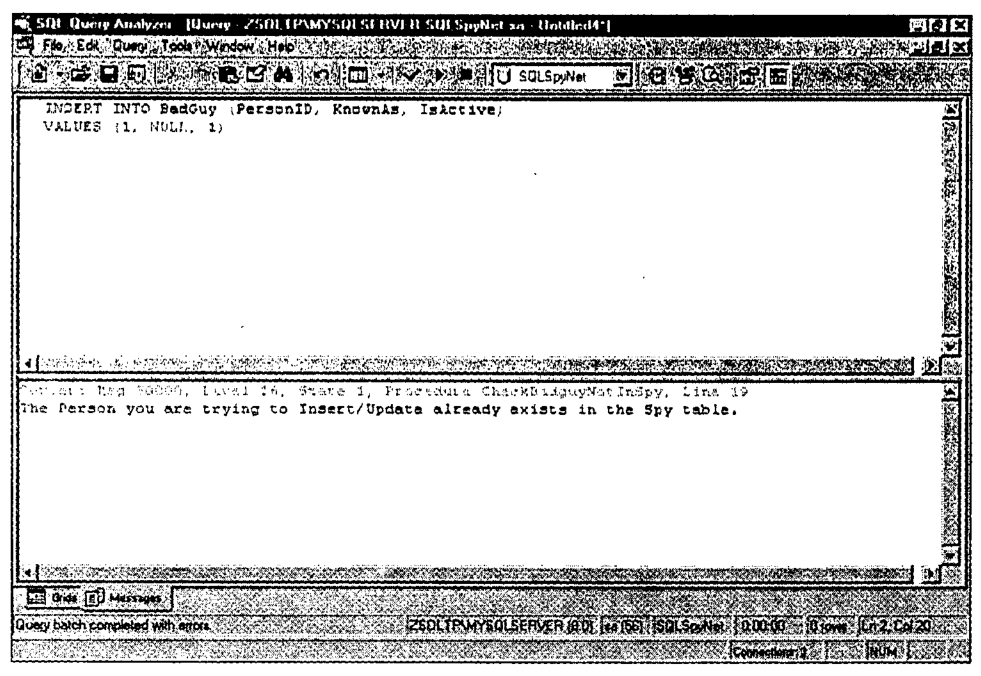


Рис. 5.6. SQL Server 2000 возвращает сообщение об ошибке, определенное в триггере CheckBadguyNotInSpy

А теперь протестируем триггер при выполнении операции UPDATE. Введите код листинга 5.15 в окно ввода кода программы Query Analyzer.

Листинг 5.15. Попытка внесения информации о двойном агенте в таблицу BadGuy с помощью операции обновления данных UPDATE

Код для запуска

```
UPDATE BadGuy SET PersonID = 1
WHERE BadGuyID = 11
```

В приведенном коде осуществляется попытка внесения в таблицу BadGuy информации о двойном агенте с помощью операции обновления данных. В результате этой попытки SQL Server 2000 возвратит сообщение об ошибке, определенное в триггере CheckBadguyNotInSpy.

И это все! Вы только что успешно создали и протестировали первый триггер базы данных SQLSpyNet. В следующем разделе описано создание аналогичного триггера для таблицы Spy.

Создание триггера для таблицы Spy

Триггер, который сейчас будет создан, практически идентичен рассмотренному ранее триггеру CheckBadguyNotInSpy (по большому счету, единственная разница между двумя триггерами заключается в таблицах, для которых они предназначены).

Введите код листинга 5.16, после чего протестируйте триггер на работоспособность с помощью выполнения приведенных далее фрагментов кода.

Листинг 5.16. Создание триггера для таблицы Spy для выявления двойных агентов

Код для запуска

```
CREATE TRIGGER CheckSpyNotInBadguy
ON Spy
FOR INSERT, UPDATE
AS
   BEGIN TRANSACTION
   DECLARE @BadGuyID INT
   DECLARE @PersonID INT
   SELECT @PersonID = i.PersonID
   FROM inserted i
    SELECT @BadGuyID = BadGuyID
    FROM BadGuy B
    WHERE B.PersonID = @PersonID
    IF (@BadGuyID IS NOT NULL) AND (@BadGuyID > 0)
        BEGIN
            RAISERROR ('The Person you are trying to
     Insert/Update already exists in the BadGuy
table.', 16, 1)
            ROLLBACK TRANSACTION
        END
    ELSE
        BEGIN
            COMMIT TRANSACTION
        END
```

Для того чтобы проверить работоспособность только что созданного триггера, выполните код, приведенный в листинге 5.17. Поскольку таблицы BadGuy и Spy уже содержат нужную информацию, не будем проводить предварительную подготовку этих таблиц путем внесения в них новых данных.

Листинг 5.17. Попытка внесения информации о двойном агенте в таблицу Spy

Код для запуска

```
INSERT INTO Spy (PersonID, AnnualSalary, IsActive)
VALUES (3, 97500.00, 1)
```

В результате выполнения приведенного кода SQL Server 2000 возвратит сообщение об ошибке, изображенное на рис. 5.7.

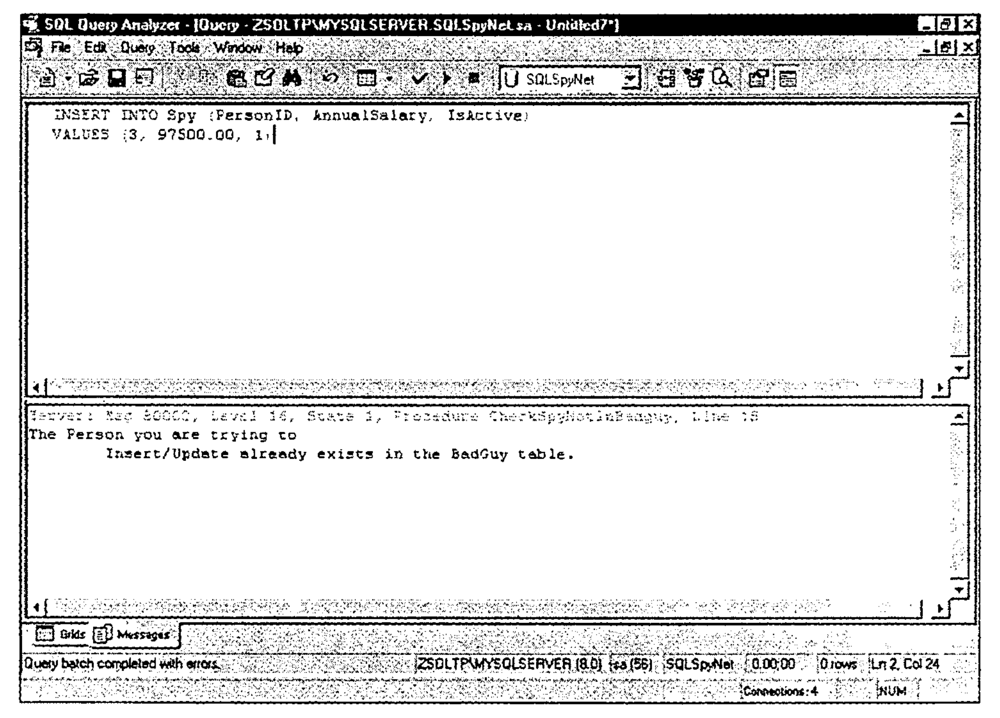


Рис. 5.7. SQL Server 2000 возвращает сообщение об ошибке, определенное в триггере `CheckSpyNotInBadguy`

А теперь протестируйте триггер `CheckSpyNotInBadguy` при выполнении операции `UPDATE`, используя для этого код, приведенный в листинге 5.18.

Листинг 5.18. Попытка внесения в таблицу `Spy` информации о двойном агенте с помощью операции обновления данных `UPDATE`

Код для запуска

```
UPDATE Spy SET PersonID = 3
WHERE SpyID = 2
```

Итак, только что вы убедились в работоспособности двух первых созданных триггеров. Проделанная работа позволяет больше не волноваться относительно нарушения целостности информации вследствие ввода данных об одном и том же человеке одновременно в таблицы `Spy` и `BadGuy`.

Удаление объектов базы данных

Что делать в том случае, когда объект языка определения данных становится больше не нужным? Наверняка вы успели отметить, что все рассмотренные в этой главе объекты языка определения данных используют при создании одинаковый

синтаксис: CREATE *ТипОбъекта ИмяОбъекта*. Подобное утверждение распространяется и на случай удаления объектов языка определения данных, с той лишь разницей, что вместо ключевого слова CREATE здесь используется ключевое слово DROP (удаление объекта языка определения данных уже рассматривалось в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet").

Перейдем к примерам! Предположим, что больше не понадобятся услуги представления, созданного в начале главы.



Не выполняйте приведенный ниже код до тех пор, пока не будете полностью уверены в необходимости удаления представления из базы данных.

Для того чтобы удалить представление, воспользуйтесь кодом, приведенным в листинге 5.19.

Листинг 5.19. Удаление ненужных объектов базы данных с помощью оператора DROP

```
1: DROP VIEW PersonAddress
```

В результате выполнения приведенного кода представление PersonAddress будет удалено из базы данных. Аналогичным образом из базы данных можно удалить практически любой ее объект. Для этого нужно всего лишь воспользоваться оператором DROP, передав ему в качестве параметров тип и имя объекта базы данных. Все гениальное просто.

Следует отметить, что все объекты базы данных поддерживают возможность их изменения. Тем не менее, вместо того чтобы изменить объект, рекомендую сначала удалить его (воспользовавшись оператором DROP), а затем создать заново.

Синтаксис оператора ALTER имеет много общего с оператором DROP. Каков же принцип его работы? По аналогии с изменением информации в базовых пользовательских таблицах при изменении объекта SQL Server 2000 сначала удаляет этот объект, а затем создает его заново.

Несмотря на то что удаление и повторное создание объекта требует от разработчика более пристального внимания, такой способ изменения объекта базы данных всегда гарантирует получение ожидаемого результата.

Следующий раздел этой главы посвящен краткому описанию одного достаточно интересного объекта языка определения данных.

Использование курсоров

В последнем разделе этой главы кратко рассматриваются курсоры — еще один объект языка определения данных. Поскольку на практике курсоры используются не так уж часто, а также потому, что приложение SQLSpyNet и вовсе не требует никаких курсоров, данная тема рассматривается здесь весьма поверхностно.

Несмотря на то что курсоры пользуются значительно меньшей популярностью, чем другие объекты базы данных, я все же настоятельно рекомендую познакомиться с принципами их работы. В некоторых случаях курсоры позволяют добиться намного большей гибкости при обращении и обновлении хранящейся в базе данных информации, нежели использующиеся для этого стандартные методы (следует также отметить, что в некоторых случаях использование курсоров может привести к значительно большим временным затратам на выполнение операции).

Что же такое курсор? При использовании хранимых процедур, представлений или каких-либо других способов доступа к информации все выполняемые при этом действия SQL Server 2000 проводит над так называемым набором данных. В 90% случаев это полностью удовлетворяет большинству требований, возникающих в ходе эксплуатации базы данных. Курсоры же необходимы в оставшихся 10% случаев.

Курсор позволяет просматривать таблицу (представление и т.п.), обращаясь отдельно к каждой ее строке. Это, в частности, подразумевает возможность выполнения операции над одной записью таблицы и при необходимости переход к ее следующей записи.

Если вы программировали в Microsoft Access (или писали программы на каком-нибудь другом языке), то, вероятно, знакомы с концепцией набора записей. Так вот, "набор записей" и "курсор" — аналогичные понятия, которые представляют собой (хотя и разными именами) один и тот же объект.

При объявлении курсор привязывается к оператору извлечения данных SELECT. В качестве источника информации для оператора SELECT может выступать представление или даже отдельная таблица. Однако еще более полезным свойством курсора является возможность встраивания его в другой курсор. Подобно хранимой процедуре, курсор может содержать операторы языка управления данными, управляющие операторы и даже операторы языка определения данных.

Итак, вы уже можете представить себе всю ту гибкость, которая достигается при использовании курсоров. Рассмотрим теперь их основные недостатки. При извлечении из базы данных достаточно большого объема информации (например, таблицы в 100 тыс. строк) с применением стандартных методов понадобится всего лишь одно обращение к базе данных, в то время как выполнение этой же операции с использованием курсоров займет время, прямо пропорциональное количеству извлекаемых из базы данных строк. Таким образом, использовать курсор для обработки больших объемов информации абсолютно нецелесообразно.

Имеют ли курсоры вообще какое-либо практическое применение? Вне всякого сомнения, да. Однажды я разрабатывал систему, которая должна была выполнять архивирование данных. Система имела очень сложные и запутанные ограничения и основывалась на использовании трех таблиц. На первый взгляд может показаться, что ничего страшного в этом нет, но поверьте, столкнуться с подобной структурой я не пожелал бы даже своему злейшему врагу.

В конце концов справиться с этой задачей мне помог курсор, который был открыт над родительской таблицей и просматривал все дочерние записи в подчиненных таблицах. В рамках приложения SQLSpyNet аналогом этому может быть открытие курсора над таблицей Person (родительская) и просмотр всех записей в таблице Address (дочерняя). В данном случае курсор предоставил мне необходимую функциональность, которой я не смог добиться использованием стандартных методов.

На этом краткое рассмотрение курсоров можно считать завершенным. Несмотря на то что при разработке приложения SQLSpyNet так и не будет создано ни одного курсора, я уверен, что вы еще успеете изучить их в ближайшем будущем. (Да что там уверен! Я гарантирую это!)

Резюме

В этой главе описано несколько ключевых операторов языка определения данных. Большинство из них будут повседневно использоваться при разработке или администрировании баз данных. И помните: ничто не в силах заменить практический опыт, а потому не бойтесь экспериментировать — это только расширит и закрепит полученные знания.

Несмотря на то что все предлагаемые SQL Server 2000 операторы определения данных не были рассмотрены, заложен крепкий фундамент, опираясь на который можно хоть сейчас начинать разработку собственного приложения!

Следующие шаги

В следующей главе описываются встроенные функции SQL Server 2000 (некоторые из них уже встречались в данной главе), а также обсуждаются вопросы применения пользовательских функций (в том числе создание собственной функции). Уверен, что вы уже в неописуемом восторге! Не так ли?

Глава 6

Использование функций для повышения эффективности управления информацией

В этой главе...

Одним из способов манипулирования данными в большинстве приложений является использование специально предназначенных для этого функций. Например, в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", рассматривалось использование при разработке приложения SQLSpyNet функции GETDATE(), возвращающей в качестве результата текущее значение даты на сервере. Наиболее существенное преимущество — возможность многократно использовать одну и ту же функцию для выполнения определенной задачи. Более того, одна и та же функция может быть использована в разных приложениях! К счастью, SQL Server 2000 поставляется с множеством встроенных функций, предназначенных для выполнения большинства часто встречающихся задач. Естественно, все это предполагает сокращение кода при значительном повышении его эффективности; таким образом, создав хорошую функцию, обязательно сохраните ее для последующего использования.

В данной главе рассматриваются некоторые функции, встроенные в SQL Server 2000, а также предлагается создать в приложении SQLSpyNet несколько собственных функций, предназначенных для манипулирования денежными суммами, датами, именами и т.д.

Прежде чем рассматривать конкретные функции, необходимо изучить общие принципы их работы, а также познакомиться с доступными типами функций. Кроме этого, в данной главе рассматривается способ совместного использования функций для выполнения сложных операций.

Роль функций в приложении

Функция — это откомпилированный набор операторов Transact-SQL, который принимает одно (либо ни одного) или целый набор значений в качестве входных параметров, возвращает значение (набор значений) или же выполняет некоторое действие. Функции позволяют выполнять одну и ту же операцию многократно, не переписывая каждый раз заново соответствующий ей код.

Вместе с SQL Server 2000 поставляется множество так называемых встроенных функций (как правило, это функции, предназначенные для выполнения стандартных операций). При желании можно создать собственные функции, называемые также пользовательскими, которые будут выполнять нужные операции или объединять встроенные функции SQL Server 2000 в более крупные наборы операторов.

Подобно хранимой процедуре, функция (как встроенная, так и пользовательская) является объектом, существующим внутри базы данных SQL Server 2000. Таким образом, функция состоит из двух частей: откомпилированного кода, создающего объект, и кода, выполняющего этот объект.

При каждом выполнении функции ей можно передавать различные параметры и вследствие этого рассчитывать на получение разных результатов ее выполнения.

На заметку Следует отметить, что встроенные и пользовательские функции имеют одно существенное различие. Поставляемые с SQL Server 2000 встроенные функции нельзя модифицировать, т.е. нельзя изменить откомпилированный код функции, для того чтобы внести поправки в выполняемую ею операцию. Несмотря на этот недостаток, встроенные функции могут быть вызваны непосредственно из кода Transact-SQL.

Код функции представляет собой комбинацию операторов определения данных, операторов управления данными и управляющих операторов. Это делает функцию столь же гибким инструментом управления данными, как хранимые процедуры и триггеры вместе взятые.

На заметку Функции SQL Server 2000 можно рассматривать как аналог функций в большинстве распространенных языков программирования. Например, в Visual Basic есть функция `Now()`, которая возвращает текущую дату на компьютере. Аналогом этой функции в SQL Server 2000 является `GETDATE()`, которая, как и функция `Now()`, возвращает значение текущей даты.

Функции могут манипулировать параметрами, переданными либо с помощью кода Transact-SQL, либо путем пользовательского ввода, изменять их и выполнять над ними вычисления. И это еще не все! Функции могут выполнять запросы к операционной системе и получать от нее важную информацию о компьютере, на котором установлен SQL Server 2000; одной из таких функций, например, является `GETDATE()` (см. главу 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet").



Функции могут быть вложены в другие функции. Одно из преимуществ такого подхода — возможность использовать внутри функции значения, полученные в результате выполнения вложенных в нее функций. Более того, вызовы функций можно размещать внутри операторов `SELECT`, в триггерах, хранимых процедурах и представлениях. Для того чтобы определить необходимость выполнения той или иной функции, обычно используются управляющие операторы.

Функции могут быть частью операторов определения данных. Например, в качестве значения по умолчанию для какого-нибудь столбца таблицы может использоваться функция `GETDATE()`. При внесении в такую таблицу новой записи в ее соответствующий столбец помещается значение текущей даты на сервере.

Экскурс ***Классификация типов функций по возвращаемому функцией значению***

Существует множество различных типов функций, которые определяются видом выполняемой операции. Однако, несмотря на это, функции всегда являются либо *детерминированными*, либо *недетерминированными*.

Термин *Детерминированные* (*deterministic*) функции всегда возвращают один и тот же результат на одинаковом наборе параметров. Например, складывая одни и те же три числа, мы всегда будем получать одинаковый результат.

Термин *Недетерминированные* (*non-deterministic*) функции могут возвращать различный результат на одном и том же наборе параметров. Например, при извлечении информации из одних и тех же строк таблицы мы вполне можем получить несколько различных результатов, отражающих внесенные в эту таблицу изменения.

Итак, все функции SQL Server 2000 являются либо детерминированными, либо недетерминированными. Несмотря на то что они могут возвращать каждый раз различные значения, недетерминированные функции выполняют как правило достаточно широкий диапазон операций.

Использование встроенных функций

Как упоминалось ранее, SQL Server 2000, а также предыдущие версии SQL Server предоставляют множество встроенных функций, которые позволяют манипулировать переданными им параметрами, проводить над ними вычисления и извлекать значения. Все эти функции чрезвычайно полезны, так как с их помощью можно реализовать дополнительные возможности приложения базы данных и более эффективно выполнять операции по администрированию базы данных.

Специально предназначенные для этого встроенные функции позволяют выполнять запросы к операционной системе; к ним, например, относится `GETDATE()`, которая будет использована вскоре в качестве составной части одной из пользовательских функций приложения SQLSpyNet.

Итак, с чего же начнем? Прежде всего, рассмотрим несколько распространенных встроенных функций SQL Server 2000 и используем их в разрабатываемом приложении.

Делай деньги с помощью функции CONVERT!

Встроенная функция SQL Server 2000 `CONVERT` принимает в качестве входного параметра данные одного типа и возвращает эти же данные, но уже в качестве другого типа. Например, можно воспользоваться функцией `CONVERT` для преобразования типа данных `INT` в `VARCHAR`.

С помощью функции `CONVERT` можно повлиять на способ объединения данных различных типов. Например, при объединении строки и числового значения SQL Server 2000 попытается преобразовать эту строку в числовой формат. В конечном итоге преобразование строки в число завершится сообщением об ошибке. Используя функцию `CONVERT`, можно добиться вполне корректного результата в описанной ситуации.

Звучит неплохо, но как воспользоваться этой замечательной функцией? Смею заверить, у вас уже есть подобный опыт. При обсуждении цикла `WHILE` в главе 4,

"Обработка данных с помощью кода Transact-SQL", функция CONVERT использовалась для установки типа переменной @MyCounter в CHAR длиной 2 символа.

В рассматриваемом ниже примере используем функцию CONVERT для того, чтобы добавить знак доллара ($) в начало каждого значения из столбца AnnualSalary таблицы Spy. Но для начала попробуем выполнить эту операцию без использования функции CONVERT, как показано в листинге 6.1.

Листинг 6.1. Попытка изменения внешнего вида столбца AnnualSalary без использования функции CONVERT

Код для запуска

```
SELECT
    SpyID,
    PersonID,
    Alias,
    DateCommencedWork,
    '$' + AnnualSalary AS Salary
FROM Spy
```

В результате выполнения приведенного кода SQL Server 2000 сгенерирует сообщение об ошибке (рис. 6.1).

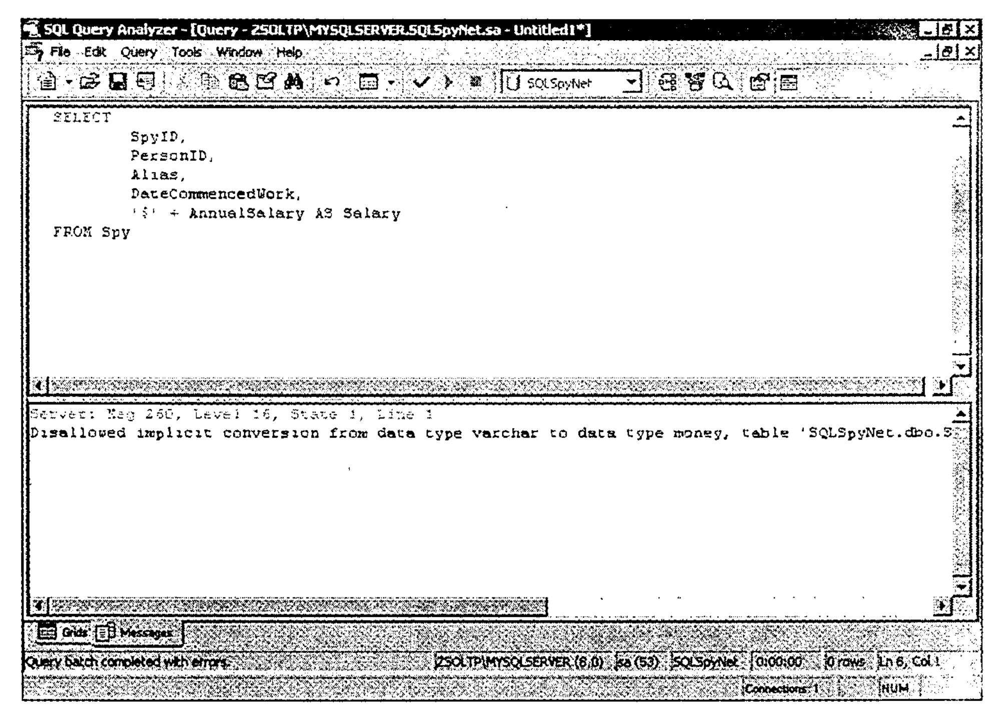


Рис. 6.1. SQL Server 2000 генерирует сообщение об ошибке в результате неявного вызова функции CONVERT

Рассмотрим, что же произошло. SQL Server 2000 пытается преобразовать символ '$' в тот тип данных, который имеет столбец AnnualSalary таблицы Spy. Операция такого типа более известна под названием *неявное преобразование*, что подразумевает выполнение операции преобразования от вашего имени, но без вашего ведома. При этом, как вы уже догадались, явного указания функции CONVERT не требуется.

Термин *Неявное преобразование* (*implicit conversion*) — это преобразование, выполняемое SQL Server 2000 от вашего имени. Если все пройдет успешно, то вы даже и не узнаете о проведении операции преобразования. Неплохо, правда?

При объединении некоторых типов данных (например, DATETIME и MONEY) SQL Server 2000 выполняет операцию неявного преобразования автоматически. Таким образом, в этой ситуации вам не стоит заботиться о явном указании вызовов функции CONVERT. Однако в некоторых случаях, например при объединении строки и числового значения, явное преобразование типов данных обязательно. Введите код из листинга 6.2 в окно ввода кода программы Query Analyzer.

Листинг 6.2. Явное преобразование числового значения в строку

Код для запуска

```
SELECT
    SpyID,
    PersonID,
    Alias,
    DateCommencedWork,
    '$' + CONVERT(VARCHAR(10), AnnualSalary) AS Salary
FROM Spy
```

В результате выполнения приведенного кода SQL Server 2000 преобразует данные столбца AnnualSalary в тип VARCHAR длиной 10 символов, что немедленно скажется на итоге выполнения всего оператора SELECT (рис. 6.2).

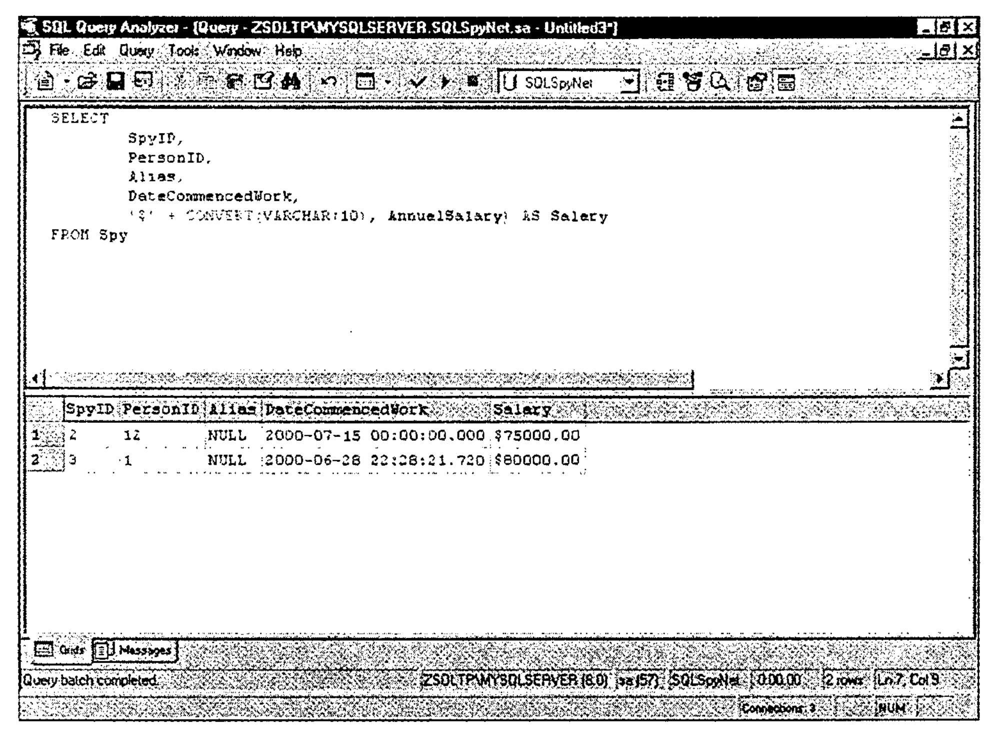


Рис. 6.2. SQL Server 2000 выполняет явное преобразование типов данных

Анализ Обратите внимание на несколько изменившийся формат вывода хранящейся в столбце AnnualSalary информации: теперь перед каждой цифрой, которая обозначает заработную плату тайного агента, стоит знак доллара ($). Не правда ли, функция CONVERT — отличная штука?

Рассмотрим строительные блоки, из которых состоит эта функция. Итак, в начале указывается само имя функции — CONVERT, за которым следует открывающая круглая скобка. Затем идет имя типа данных, в который необходимо преобразовать нужное значение. В рассматриваемом примере таким типом данных является VARCHAR(10).

Далее указывается значение, тип которого необходимо изменить. Это может быть имя столбца, как в приведенном выше примере, или же конкретное значение, например CONVERT(VARCHAR(2), 22). Обратите внимание, что число 22 берется не из какого-либо внешнего источника, а помещается непосредственно в самом вызове функции.

В результате выполнения приведенного выше примера вызова функции CONVERT тип данных значения 22 будет изменен с числового на строковый (VARCHAR), таким образом 22 превратится в '22'.

И последнее: вызов функции CONVERT оканчивается закрывающей круглой скобкой.

Итак, дамы и господа, только что мы кратко описали основные особенности функции CONVERT. Ах да, извините, совсем забыл упомянуть о функции CAST! Она практически аналогична функции CONVERT и тоже позволяет изменять тип переданного ей значения. Однако CONVERT имеет одно преимущество перед функцией CAST: наличие дополнительного параметра style.

На заметку Функция CAST обладает схожей функциональностью с CONVERT, однако способна принимать меньшее по сравнению с CONVERT количество параметров.

Параметр style указывается последним из параметров функции CONVERT и позволяет задать "стиль", который будет использован при преобразовании типов данных DATETIME, SMALLDATETIME и большинства числовых типов в строковые типы данных (VARCHAR, CHAR, NVARCHAR и т.д.).

С учетом параметра style можно изменить строку 6 рассмотренного выше примера, как показано в листинге 6.3.

Листинг 6.3. Использование параметра style функции CONVERT для изменения стиля возвращаемого значения

Код для запуска

```
1: SELECT
2:     SpyID,
3:     PersonID,
4:     Alias,
5:     DateCommencedWork,
6:     '$' + CONVERT(VARCHAR(10), AnnualSalary, 1) AS
6a: Salary
7: FROM Spy
```

Результат выполнения приведенного выше кода практически аналогичен предыдущему результату, за исключением того, что в столбце Salary для отделения разрядов чисел теперь используются запятые (,), как показано на рис. 6.3.

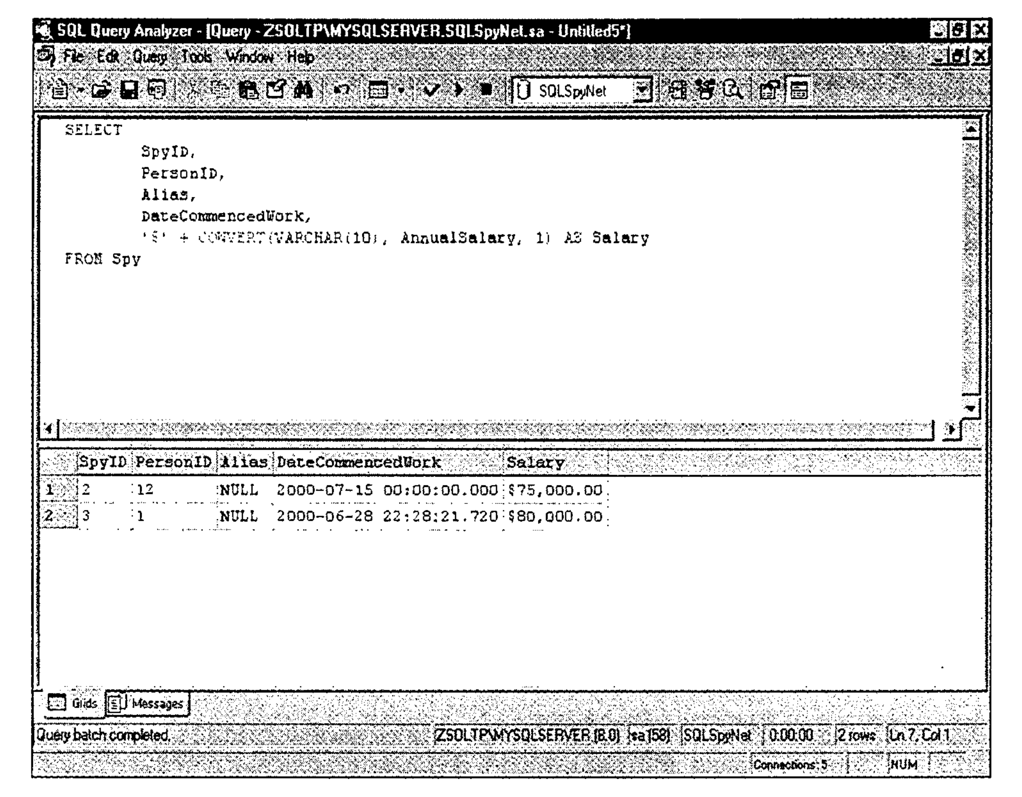


Рис. 6.3. SQL Server 2000 выполняет явное преобразование типов данных с применением базового стилевого форматирования

Параметр style позволяет задать внешний вид отображаемых данных. Не правда ли, очень удобный параметр?

Именно из-за этой мелкой, но достаточно приятной дополнительной особенности функции CONVERT я предпочитаю использовать ее (а не функцию CAST) для преобразования типов данных. Разумеется, ваш личный выбор между функциями CAST и CONVERT должен базироваться только на собственных предпочтениях.

Одной из наиболее распространенных встроенных функций SQL Server 2000 является COUNT, которой посвящен следующий раздел этой главы.

Подсчет количества тайных агентов с помощью функции COUNT

Как следует из названия (*count* — считать), функция COUNT предназначена для проведения подсчетов. В качестве результата своей работы эта функция возвращает общее количество элементов в некотором множестве. Тип возвращаемых функцией COUNT данных — INT (целое число).

Функция COUNT относится к так называемым *обобщающим* функциям, поскольку производит подсчет количества элементов. Еще одной ее характеристикой является детерминированность: функция COUNT всегда возвращает один и тот же результат при неизменных входных параметрах.

Термин Как правило, *обобщающие функции* (*aggregate functions*) используются для проведения операций над числовыми данными и измерений каких-либо характеристик. Более подробно обобщающие функции описываются далее в главе.

В большинстве случаев функция COUNT применяется для определения общего количества строк в таблице. Например, для того чтобы узнать, сколько тайных агентов занесено в таблицу Spy, следует всего лишь воспользоваться функцией COUNT и получить ответ.

Функция COUNT может быть встроена в операторы Transact-SQL, включая операторы определения данных и операторы управления данными. Таким образом, эту функцию можно использовать в хранимых процедурах, триггерах, представлениях, а также в операторах SELECT, INSERT и UPDATE.

Прежде чем применить функцию COUNT на практике, необходимо уяснить несколько моментов, непосредственно касающихся ее использования:

- если все элементы множества, по отношению к которому будет применена функция COUNT, имеют значения NULL, то в результате своей работы эта функция возвратит ноль (0);
- для того чтобы применить функцию COUNT по отношению ко всем элементам таблицы, следует воспользоваться ключевым словом ALL (оно указывает функции COUNT на необходимость включения в результат подсчета всех значений заданного столбца таблицы; такое "поведение" функции COUNT является заданным по умолчанию);
- вместе с функцией COUNT можно использовать еще одно ключевое слово — DISTINCT (оно указывает функции COUNT на необходимость включения в результат подсчета только уникальных элементов заданного множества);
- наконец, можно использовать символ звездочки (*) для указания функции COUNT на необходимость проведения подсчета над всеми *столбцами* и *строками* таблицы.

Рассмотрим небольшой пример, использовав функцию COUNT для подсчета общего количества записей в таблице Person.

Прежде всего необходимо создать оператор SELECT, который будет использовать функцию COUNT для того, чтобы возвратить значение, равное общему числу записей в таблице Person (листинг 6.4).

Листинг 6.4. Подсчет общего количества записей в таблице Person

Код для запуска

```
SELECT COUNT(*) AS TotalPeople FROM Person
```

В результате выполнения приведенного кода SQL Server 2000 возвратит значение, равное общему количеству строк в таблице Person. При необходимости можно повлиять на это значение, применив уже известный вам критерий WHERE (употребление оператора SELECT вместе с функцией COUNT, по сути, ничем не отличается от обычного применения этого оператора). Также можно провести объединение таблицы Person с другими таблицами базы данных SQLSpyNet.

Этим применение функции COUNT не ограничивается. Как упоминалось ранее, можно воспользоваться ключевым словом DISTINCT для того, чтобы указать функции COUNT на необходимость подсчета только уникальных элементов множества.

В какой же таблице базы данных SQLSpyNet есть повторяющиеся значения? Естественно, в таблице Address. В результате выполнения кода, приведенного в листинге 6.5, SQL Server 2000 возвратит общее число идентификационных номеров PersonID, содержащихся в таблице Address.

Листинг 6.5. Подсчет числа значений PersonID в таблице Address

Код для запуска

```
1: SELECT COUNT(PersonID) FROM Address
```

В результате выполнения кода, приведенного в листинге 6.6, SQL Server 2000 возвратит общее число *уникальных* идентификационных номеров PersonID, содержащихся в таблице Address.

Листинг 6.6. Подсчет числа уникальных значений PersonID в таблице Address

Код для запуска

```
1: SELECT COUNT(DISTINCT PersonID) FROM Address
```

Вот и все! Уверен, что вы не испытали ни малейших трудностей при изучении функции COUNT. Однако, прежде чем переходить к созданию собственной функции, вы должны познакомиться еще с двумя встроенными функциями SQL Server 2000. Итак, поехали!

Использование функции SUM для подсчета выплат по заработной плате

Подобно COUNT, функция SUM является обобщающей детерминированной функцией. Она позволяет узнать сумму чисел, входящих в заданное множество (как правило, это множество представляет собой столбец таблицы с числовыми значениями или же просто набор чисел). Подобно COUNT, функция SUM возвращает результат в виде значения, тип которого наиболее соответствует типу исходных данных. Это может быть тип INT, FLOAT, MONEY и т.д.

Для чего же предназначена функция SUM? Те, кто использовал ранее программу Excel, наверняка уже догадываются о потенциальной мощи этой функции; те же, кто никогда не работал с Excel, скоро о ней узнают!

Рассмотрим пример, в котором функция SUM используется для подсчета общей суммы денег, выплачиваемых в качестве заработка тайным агентам. Введите код листинга 6.7 в окно ввода кода программы Query Analyzer.

Листинг 6.7. Ничего себе! Эти бездельники скоро разорят нас!

Код для запуска

```
1: SELECT SUM(AnnualSalary) FROM Spy
```

В результате выполнения приведенного кода SQL Server 2000 просуммирует все значения, находящиеся в столбце AnnualSalary таблицы Spy, и возвратит полученное число. Как и в случае с функцией COUNT, можно воспользоваться ключевым словом DISTINCT для того, чтобы указать функции SUM необходимость включения в результат только уникальных значений заданного столбца. Следует отметить, что проявить все свои лучшие качества функции SUM помогает другая обобщающая функция — AVG, которая возвращает среднее значение для заданного множества чисел.

Итак, воспользуемся обеими функциями для того, чтобы сгенерировать наглядный отчет, позволяющий получить представление о суммах, выплачиваемых в качестве заработка тайным агентам (листинг 6.8).

Листинг 6.8. Вы только посмотрите, сколько они получают в среднем!..

Код для запуска

```
SELECT
    SUM(AnnualSalary) AS TotalSalaries,
    AVG(AnnualSalary) AS AverageSalary
FROM Spy
```

В результате выполнения приведенного кода SQL Server 2000 возвратит не только общую сумму денег, выплачиваемых тайным агентам, но еще и среднее значение заработной платы рядового агента (рис. 6.4).

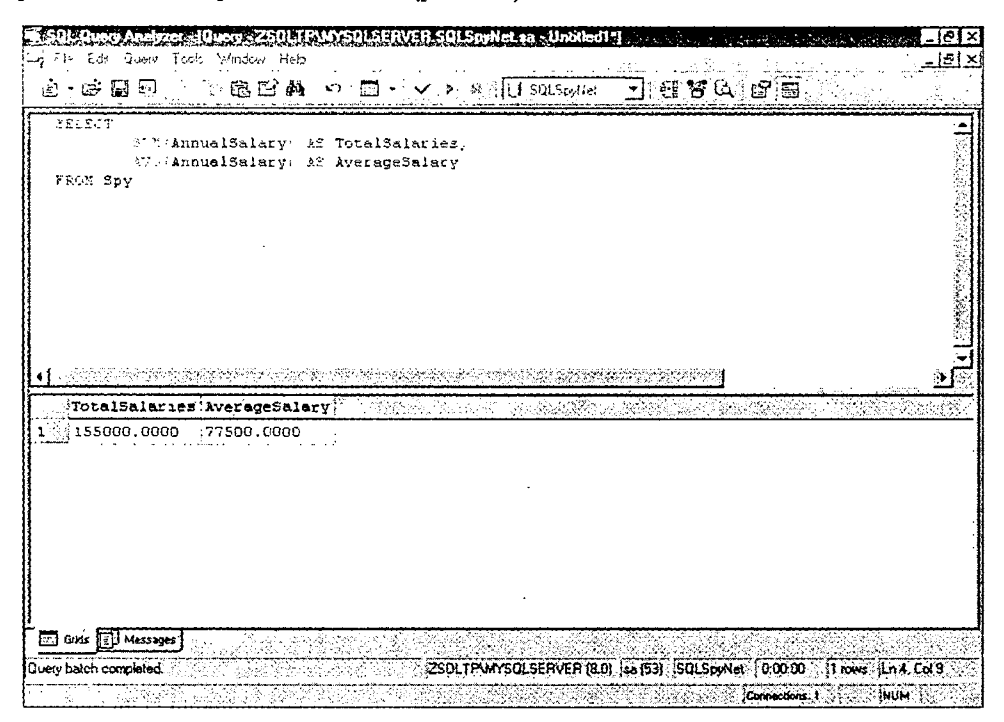


Рис. 6.4. SQL Server 2000 возвращает не только общую сумму, выплачиваемую тайным агентам, но и заработную плату "среднестатистического" агента

Имея на руках информацию о заработной плате среднестатистического тайного агента, можно быстро вычислить тех, кто больнее всего бьют по бюджету нашего маленького тайного агентства!

Итак, на текущий момент уже рассмотрены две (а на самом деле даже три) обобщающие функции. Тем не менее, прежде чем приступить к созданию собственной функции, необходимо познакомиться еще с одной встроенной функцией SQL Server 2000.

Использование функции STUFF для форматирования строк

Рассмотрим скалярную функцию STUFF, которая позволяет достаточно гибко форматировать строковые значения. Благодаря этой функции можно заменить набор определенных пользователем символов строки новым набором символов (естественно, определенных пользователем), который будет размещен в строке в заданной пользователем позиции.

Термин *Скалярные функции* (*scalar functions*) — это функции, которые оперируют одним-единственным значением и возвращают в качестве результата также одно-единственное значение. К скалярным относится большинство конфигурационных функций (т.е. возвращающих информацию о настройках сервера). Более подробно скалярные функции описываются далее в главе.

Несмотря на то что STUFF уступает рассмотренным ранее обобщающим функциям в частоте использования, она, бесспорно, остается очень мощным инструментом, позволяющим легко изменить внешний вид строкового значения.

Введите код листинга 6.9 в окно ввода кода программы Query Analyzer.

Листинг 6.9. Изменение внешнего вида строки с помощью функции STUFF

Код для запуска

```
SELECT STUFF(Firstname, 1, 0, 'A Person named ')
FROM Person
```

SQL Server 2000 разместит строку 'A Person named ' в начале каждой строки из столбца Firstname. Результат выполнения приведенного выше кода представлен на рис. 6.5.

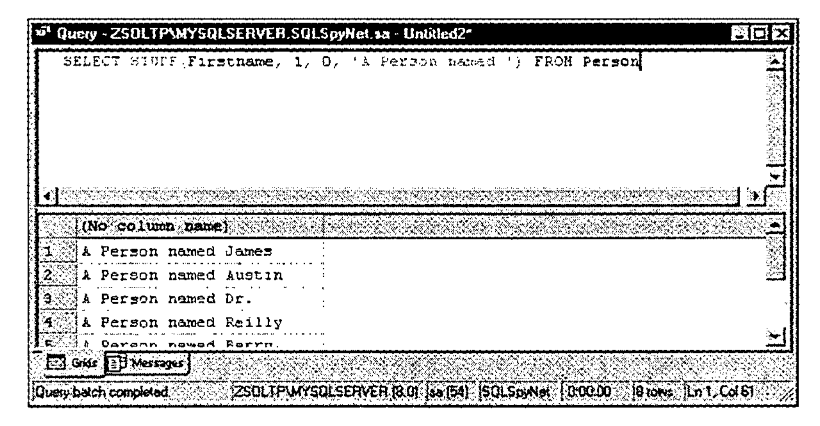


Рис. 6.5. SQL Server 2000 изменяет внешний вид значений столбца Firstname

Попробуем разобраться в принципе работы функции STUFF. Формат этой функции (ее синтаксис) определяется принимаемыми параметрами:

STUFF(*Строка_в_которой_будет_проведена_замена*, *Начало*, *Длина*, *Подставляемое_в_строку_значение*)

Первый аргумент функции STUFF — это строка, внешний вид которой необходимо изменить (в данном случае это значения столбца Firstname).

Второй аргумент позволяет определить позицию внутри строки, куда затем будет вставлено новое значение. В приведенном выше примере этот аргумент равен 1, что обозначает наше намерение вставить новое значение в самое начало изменяемой строки.

Третий аргумент функции STUFF определяет число элементов, которые должны быть удалены из исходной строки. В рассматриваемом примере это значение равно 0, т.е. мы не собираемся удалять из исходной строки ни одного элемента. Если бы вместо нуля было указано любое положительное число, то именно столько символов было бы "затерто" при вставке в строку нового значения.

Четвертый, и последний, аргумент функции STUFF позволяет задать новое значение, которое будет вставлено в исходную строку.

На заметку Стоит отметить, что изменения, вносимые с помощью функции STUFF в хранящуюся в базе данных информацию, не являются постоянными. Для внесения постоянных изменений необходимо воспользоваться оператором UPDATE, применив его вместе с функцией STUFF.

Ввиду временного характера изменений, вносимых с помощью функции STUFF, попробуйте поэкспериментировать со значениями начальной позиции и количества замещаемых символов строки.

Подведем небольшой итог. В предыдущих разделах этой главы вы познакомились с несколькими встроенными функциями SQL Server 2000, включая обобщающие и скалярные. Смею заверить, что этих знаний вполне достаточно, чтобы приступить к самому интригующему моменту данной главы — созданию собственной функции.

Создание собственных функций, предназначенных для манипулирования данными

Несмотря на то что SQL Server 2000 обладает достаточно широким набором различных встроенных функций, неизбежно наступает время, когда мы начинаем понимать, что стандартных функций не хватает для решения конкретных задач разрабатываемого приложения.

Что же делать в подобной ситуации? Ответ прост: создать собственные функции. Создание собственных функций позволяет разрабатывать достаточно мощные приложения, манипулирующие информацией на уровне базы данных. Еще одно серьезное преимущество использования собственных функций — возможность их быстрого изменения в ответ на изменение требований к приложению.

Означает ли это, что в ходе освоения материала следующего раздела вы сможете создать свою первую функцию? Ну конечно, да! SQL Server 2000 предоставляет возможность разработки небольших фрагментов кода, которые вызываются из Transact-SQL и могут возвращать значение в зависимости от переданных им параметров.

На заметку Как и в случае с большинством других объектов базы данных, создание функции представляет собой двухэтапный процесс. Первый этап — непосредственное создание функции с помощью оператора определения данных, второй — вызов созданной функции с помощью операторов управления данными.

Большинство пользовательских функций скалярные; они возвращают одно-единственное значение в результате обработки одного или многих параметров. По отношению к функциям может быть применена практика ограничения прав на их использование. Таким образом, и в этой области SQL Server 2000 предоставляет возможность защиты важной информации. Более подробно соображения, касающиеся безопасности базы данных, рассматриваются в главе 7, "Защита пользовательского ввода от возможных ошибок с помощью стандартных значений и правил".

Как упоминалось ранее, внутри пользовательских функций могут размещаться вызовы встроенных функций SQL Server 2000. Таким образом можно расширить сферу применения некоторых функций SQL Server 2000 и приспособить их к конкретным требованиям разрабатываемого приложения.

Создание пользовательской функции, форматирующей дату

Одной из функций, требующих немедленной доработки, является функция, которая форматирует дату для последующего вывода на экран. Как уже упоминалось, во многих странах принят различный формат даты. Например, в Европе дата записывается как день/месяц/год, в США — как месяц/день/год. Вследствие этого обновления сервера, настроенного на формат даты США, на компьютере с европейским форматом даты становится поистине проблематичным. Например, дата 06/07/00 будет интерпретироваться как 7 июня в американском формате и как 6 июля в европейском! Что же делать?

В данном разделе рассмотрим создание функции, которая принимает дату в качестве входного параметра и возвращает ее в формате, стопроцентно гарантирующем его универсальность. Итак, введите код листинга 6.10 в окно ввода кода программы Query Analyzer.

Листинг 6.10. Создание функции, гарантирующей стопроцентную корректность формата даты

Код для запуска

```
CREATE FUNCTION DateFormatter
(@Date DATETIME, @DateSeperator CHAR(1))
RETURNS VARCHAR(20)
AS
BEGIN
  DECLARE @ReturnDate VARCHAR(20)
  SET @ReturnDate = CONVERT(VARCHAR(2), DAY(@Date)) +
    @DateSeperator +
   DATENAME(MONTH, @Date) + @DateSeperator +
   CONVERT(VARCHAR(4), YEAR(@Date))
  RETURN(@ReturnDate)
END
```

Вот и все! Вы только что создали первую пользовательскую функцию, проводящую форматирование даты. В результате SQL Server 2000 откомпилирует и выполнит соответствующий оператор определения данных и сгенерирует сообщение, подтверждающее успешное выполнение кода Transact-SQL.



Анализ

Проанализируем код Transact-SQL, приведенный в листинге 6.10.

- Расположенный в строке 1 оператор определения данных задает тип создаваемого объекта SQL Server 2000.
- В строке 1а перечисляются входные параметры функции `DateFormatter`, а также их типы. В рассматриваемом примере первым параметром функции является дата, а вторым — единичный символ.
- В строке 2 определяется тип данных, который будет возвращать создаваемая функция вызвавшей ее процедуре. Почему же вместо типа `DATETIME` указан тип `VARCHAR`? Дело в том, что в случае указания `DATETIME` в качестве типа возвращаемых данных пришлось бы неизбежно столкнуться с обратным преобразованием даты в строковом формате в тип `DATETIME`. Таким образом, фактически игнорируется вся работа функции `DateFormatter`, которая в таком случае становится просто бесполезной!
- Уверен, что остаток кода не вызовет у вас никаких затруднений. Единственное, что стоит отметить, — это использование встроенных функций SQL Server 2000 `DATENAME` и `CONVERT`.
- Последняя заслуживающая внимания строка — 9. Расположенный в ней оператор `RETURN` обозначает необходимость возврата значения переменной `@ReturnDate` (отметьте еще раз, что эта переменная имеет тип `VARCHAR`) вызывавшей функцию `DateFormatter` подпрограмме.

Как видите, процесс создания функции не очень сложен. Следует отметить, что пользовательская функция не должна быть слишком большой и запутанной. Если все же без этого не обойтись, попробуйте разбить функцию на несколько меньших частей и оформите каждую из них в виде отдельной функции. Зачем я на этом настаиваю? Попробуйте вернуться к созданной базе данных месяцев шесть спустя, и я уверен, вы будете только благодарны мне за ценный совет. К тому же необходимо добавить, что небольшие фрагменты кода выполняются и компилируются гораздо быстрее, нежели большие и запутанные.

Вызов пользовательской функции

Итак, первая функция создана, но как же ее применить на практике? Вызов пользовательской функции практически аналогичен вызову встроенной функции SQL Server 2000, отличие лишь в том, что при вызове пользовательской функции необходимо указывать имя ее "владельца". Рассматривая приведенный ниже код, вы наверняка обратите внимание на префикс `dbo`, расположенный перед именем пользовательской функции. Более подробно префикс `dbo` рассматривается при обсуждении системы безопасности базы данных.

Обратите внимание на два параметра, передаваемых в функцию `DateFormatter`. Первый из них — это встроенная функция SQL Server 2000 `GETDATE()`, второй — разделяющий символ, который будет использоваться при отображении даты на экране компьютера.

```
SELECT dbo.DateFormatter(GETDATE(), '/') AS DateNow
```

В строке кода, приведенной в листинге 6.11, функция `DateFormatter` используется вместе с оператором `SELECT`, который возвращает даты дней рождения тайных агентов и злоумышленников в универсальном формате (рис. 6.6).

Листинг 6.11. Использование функции DateFormatter для форматирования даты

Код для запуска

```
SELECT dbo.DateFormatter(DOB, '-') AS FormattedDOB
FROM Person
```

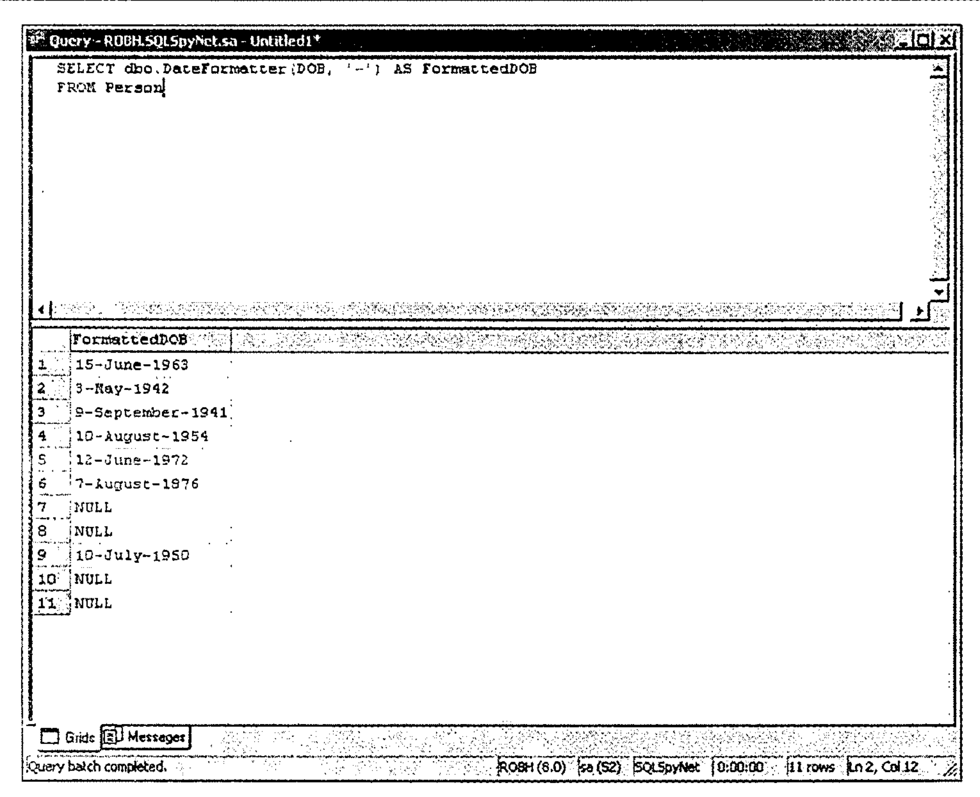


Рис. 6.6. Использование функции `DateFormatter` *для форматирования дат дней рождения тайных агентов и злоумышленников*

Учитывая структуру функции `DateFormatter`, в качестве второго параметра можно было бы указать пробел или какой-либо другой разделяющий символ. Не бойтесь экспериментировать!

Создание собственной библиотеки функций

В процессе работы с SQL Server 2000 вы начнете создавать собственную библиотеку функций. Стоит отметить, что у вас уже есть неплохая база: рассмотренные в этой главе пользовательские функции могут стать первыми "томами" такой "библиотеки". Уверен, что в будущем вы создадите еще не одну великолепную функцию, базируясь при этом на изложенных ранее основных концепциях. При этом крайне важно четкое понимание классификации функций по типам и выполняемым действиям; так что, столкнувшись с определенной проблемой, вы будете иметь луч-

шее представление о том, какая "часть" вашей библиотеки поможет разрешить данную проблему с минимальными затратами.

Ниже перечислены три основные категории функций, с которыми мы уже встречались в этой главе.

- Скалярные функции оперируют одним-единственным значением и возвращают в качестве результата работы также одно-единственное значение.
- Функции набора строк возвращают в качестве результата работы объект, который может быть использован вместо таблицы (например, функция OPENROWSET позволяет посредством кода Transact-SQL обращаться напрямую к оператору SELECT);
- Обобщающие функции предназначены для проведения операций над числовыми данными и измерений каких-либо характеристик. К ним относятся, например, функции COUNT (позволяет провести подсчет количества элементов в заданном множестве), SUM (позволяет подсчитать сумму элементов в заданном множестве) и MAX (позволяет определить максимальный элемент в заданном множестве). Все обобщающие функции принимают в качестве входного параметра множество значений, а возвращают в качестве результата работы одно-единственное.

Скалярные функции

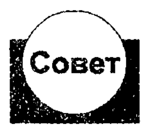
Скалярные функции, входящие в поставку SQL Server 2000, делятся на несколько основных подкатегорий:

- конфигурационные функции возвращают информацию о текущей конфигурации базы данных;
- функции для работы с временем и датами принимают в качестве входных параметров значение даты или времени и возвращают в качестве результата работы число (целое, натуральное и т.д.), строку или значение времени/даты;
- функции курсора возвращают информацию о курсорах, определенных в пределах данного экземпляра SQL Server 2000;
- функции метаданных (очень гибкие и многоцелевые) возвращают информацию о базе данных и содержащихся в ней объектах;
- математические функции принимают в качестве входных параметров числовые значения, проводят над ними определенные вычисления, и возвращают в качестве результата работы одно или несколько числовых значений;
- строковые функции принимают в качестве входных параметров строковые значения и возвращают в результате работы строковое значение или число (целое, натуральное и т.д.);
- функции безопасности позволяют получить информацию о пользователях и ролях, определенных в базе данных;
- функции для работы с текстом и изображениями принимают в качестве входных параметров текст или изображение и возвращают в результате работы информацию о значениях этих параметров;
- системные функции позволяют получить информацию об объектах базы данных и настройках SQL Server 2000;
- системные статистические функции позволяют получить статистическую информацию о текущей конфигурации системы.

Обобщающие функции и функции набора строк

А что же обобщающие функции и функции набора строк? Существуют ли и у них подкатегории, подобные подкатегориям скалярных функций? К счастью, нет. Следует отметить, что функции набора строк недетерминированные. Это означает, что результат их выполнения может быть различным при неизменных входных параметрах. Почему же это происходит? Функции набора строк возвращают в качестве результата работы строки информации, хранящейся в базе данных. Естественно, что эти строки со временем могут быть изменены, например, каким-либо пользователем базы данных, и именно поэтому невозможно гарантировать получение одинакового результата при выполнении функций набора данных на одних и тех же входных параметрах.

С другой стороны, обобщающие функции являются детерминированными. При неизменных входных параметрах они всегда возвращают одно и то же значение. Действительно, стоит лишь представить себе, что произойдет, если функция COUNT каждый раз будет возвращать различный результат, — просто кошмар какой-то!



Все обобщающие функции, за исключением COUNT, игнорируют значения NULL.

Полный список встроенных функций SQL Server 2000 можно найти в поставляемой с этой СУБД документации.

Резюме

На текущий момент вы уже должны обладать небольшим набором собственных функций, которые могут быть использованы как в SQLSpyNet, так и в последующих разрабатываемых вами приложениях. Помимо этого, не следует забывать об огромном "багаже" встроенных функций, который был "доставлен" вместе с SQL Server 2000. Используйте их вместе с функциями, созданными собственноручно (естественно, после их тщательного тестирования), для решения множества различных задач, возникающих в процессе разработки приложения базы данных.

Стоит отметить, что, поскольку функции могут быть использованы для проверки хранящейся в базе данных информации на предмет ее соответствия определенным требованиям, они автоматически становятся еще и великолепным средством обеспечения целостности информации.

Следующие шаги

В следующей главе вы познакомитесь с такими основными понятиями теории реляционных баз данных, как транзакции и блокировка. Это наиболее важные особенности всей реляционной теории, присущие работе с базой данных в многопользовательском режиме.

Глава 7

Защита пользовательского ввода от возможных ошибок с помощью стандартных значений и правил

В этой главе...

Введение

Обеспечение целостности информации — один из самых сложных этапов разработки приложения базы данных. Трудности начинаются уже на первом этапе решения этой задачи: практически все разработчики базы данных в той или иной степени уверены, что пользователи смогут без труда вносить в нее только корректную информацию. К сожалению, они глубоко заблуждаются. Большинство пользователей, в особенности новички по отношению к рассматриваемой базе данных, с завидной регулярностью пытаются внести в нее некорректную информацию. Это обусловлено, прежде всего, банальной склонностью человека к совершению непреднамеренных ошибок.

В связи с этим совершенно очевидно, что одна из основных задач разработчика приложения — свести к минимуму возможность внесения в базу данных некорректной информации, которая может негативно повлиять на ее целостность. Добиться этого можно с помощью использования так называемых ограничивающих условий целостности данных.

Ранее мы рассматривали методы поддержки целостности хранящейся в базе данных информации, основанные на использовании триггеров, хранимых процедур и представлений. Как вы понимаете, это далеко не полный арсенал средств обеспечения целостности, предлагаемых SQL Server 2000. Определить дополнительные условия целостности помогут правила, стандартные значения и пользовательские типы данных.

Итак, в этой главе рассматривается использование правил, стандартных значений и определяемых пользователем типов данных для обеспечения надежной гарантии защиты хранящейся в базе данных информации.

Для каждого из этих средств будут определены необходимые условия применения и приведен конкретный пример использования в рамках разработки приложения SQLSpyNet.

Использование правил

При создании таблиц базы данных SQLSpyNet (кажется, что с того момента прошла уже целая вечность!) мы определили несколько условий, обеспечивающих целостность информации. В частности, были реализованы правила целостности данных на уровне доменов, сущностей и ссылок между таблицами. По крайней мере в одном из сценариев был реализован и четвертый тип — *правила целостности, определяемые пользователем.*

Термин *Правила целостности, определяемые пользователем* (*user-defined integrity*), позволяют задать собственные ограничения на информацию, которая может быть внесена в столбцы таблиц.

Что же это был за сценарий? Вспомните триггер, отслеживающий ситуацию с двойными агентами: человек, которому соответствует одно и то же значение столбца PersonID, не может быть одновременно занесен в таблицы Spy и BadGuy.

Однако если у нас есть триггеры — великолепный инструмент обеспечения целостности информации, то зачем нужен еще один новый тип правил целостности? Дело в том, что в большинстве случаев триггеры применяются для реализации достаточно сложной логики программы, а также для реагирования на какое-либо событие базы данных. Для выполнения более простых действий или же для немедленного уведомления пользователя об ошибке ввода данных лучше всего воспользоваться таким средством SQL Server 2000, как правило.

В некотором смысле правило напоминает уже рассматривавшуюся ранее функцию. Это небольшой откомпилированный фрагмент кода, достаточно простой по своей природе и к тому же легко модифицируемый.

На заметку Результатом работы правила не может быть возврат какого-либо значения. Тем не менее, применив в правиле функцию RAISERROR, можно уведомить пользователя о некорректном вводе данных с помощью собственного сообщения об ошибке. Таким образом, при нарушении определенного правила пользователь сможет получить достаточно информации, разъясняющей его оплошность (например, "Вы не можете отправить в отставку агента, не достигшего 60-летнего возраста"). Правило применяется к столбцу таблицы; при этом необходимо отметить, что, к сожалению, к одному столбцу таблицы может быть применено лишь одно правило.

Выявление несовершеннолетних агентов

Предположим, что в таблице Person не должны храниться записи об агентах, не достигших 16-летнего возраста (возраст, начиная с которого во многих странах разрешается официальное трудоустройство). Естественно, это условие является одним из необходимых ограничений, накладываемых на хранящуюся в таблице Person информацию.

Итак, мы нашли ограничение, которое можно великолепно реализовать в виде правила. Почему именно это ограничение? Во-первых, оно достаточно простое для

того, чтобы воплотить его с помощью триггера. Во-вторых, оно может измениться с принятием нового закона о трудоустройстве, разрешающего, например, работать, начиная с 15-летнего возраста. В этом случае разработчику базы данных необходимо будет лишь изменить одно из значений в исходном коде соответствующего правила.



К сожалению, применяемое к столбцу таблицы правило никоим образом не влияет на уже содержащуюся в нем информацию. Поэтому, если в таблице `Person` *перед* применением правила будет находиться запись о несовершеннолетнем агенте, то применение правила не позволит выявить эту оплошность.

Итак, приступим к делу. Обратите внимание на то, что код создаваемого нами правила очень прост и состоит всего лишь из трех строк (листинг 7.1).

Листинг 7.1. Создание правила для выявления несовершеннолетних тайных агентов и злоумышленников

Код для запуска

```
1: CREATE RULE AgeValidation_Rule
2: AS
3: @DOB < DATEADD(YEAR, -16, GETDATE())
```

Анализ

Наверняка вы обратили внимание на то, что в приведенном выше правиле были использованы сразу две встроенные функции SQL Server 2000 — `DATEADD()` и `GETDATE()`. Это вполне легитимно и к тому же значительно повышает функциональность оператора `CREATE RULE`.

Единственное, что действительно заслуживает внимания, так это код строки 3 (где, собственно, и происходит все самое интересное). Переменная `@DOB` является локальной, хранящей текущее значение столбца. Имя этой переменной, вообще говоря, может быть любым, однако его всегда *должен* предварять префикс `@`.

Следующим шагом является проверка хранящегося в переменой `@DOB` значения: если оно *не* меньше 16, условное выражение возвращает "ложь". В этом случае операция вставки/обновления (`INSERT/UPDATE`) информации проходит успешно, в противном завершается неудачей.

Привязка и проверка работоспособности правила

К сожалению, мы пока не можем проверить работоспособность созданного правила. В чем же дело? Для того чтобы протестировать правило, его необходимо сначала *привязать* к соответствующему столбцу таблицы.

Термин

В рамках компьютерной терминологии *привязка правила* (*binding a rule*) — это установка логической связи правила с определенным объектом базы данных (в нашем случае со столбцом). Таким образом, "привязанное" правило связано, или ассоциировано, с определенным столбцом таблицы. Термин *привязка* широко используется в различных областях компьютерных технологий, например в сфере разработки приложений базы данных Microsoft Access.

Для того чтобы привязать правило к столбцу, можно воспользоваться либо встроенной хранимой процедурой SQL Server 2000, либо программой Enterprise Manager.

Как было отмечено выше, sp_bindrule является встроенной хранимой процедурой, поставляющейся вместе с SQL Server 2000. Она позволяет программным способом привязать правило к определенному столбцу таблицы. Ниже приведен синтаксис хранимой процедуры sp_bindrule:

```
1: sp_bindrule 'правило', 'имя_объекта', 'флаг_последующего_использования'
```

Итак, процедура sp_bindrule принимает три входных параметра.

- *правило* — имя правила, привязку которого необходимо осуществить. В нашем случае это AgeValidation_Rule.
- *имя_объекта* — имя объекта, с которым должно быть связано правило. Указывая имя столбца таблицы, следует использовать синтаксис *имя_таблицы.имя_столбца*, что в нашем случае будет выглядеть как Person.DOB.
- *флаг_последующего_использования* — необязательный параметр со значением по умолчанию, равным NULL (используется в том случае, если параметр опущен). Применяется только по отношению к пользовательским типам данных и позволяет определить способ привязки правила, состоящий в его применении исключительно к последующему использованию этого типа данных (что означает "неприкосновенность" уже существующей информации этого типа).

Привяжем созданное правило к столбцу DOB таблицы Person и проверим его работоспособность на характерном наборе вносимых в эту таблицу данных. Введите код листинга 7.2 в окно ввода кода программы Query Analyzer.

Листинг 7.2. Привязка привила AgeValidation_Rule к столбцу DOB таблицы Person

Код для запуска

```
1: EXEC sp_bindrule 'AgeValidation_Rule', 'Person.DOB'
```

В результате выполнения этого кода SQL Server 2000 возвратит сообщение, как показано на рис. 7.1.

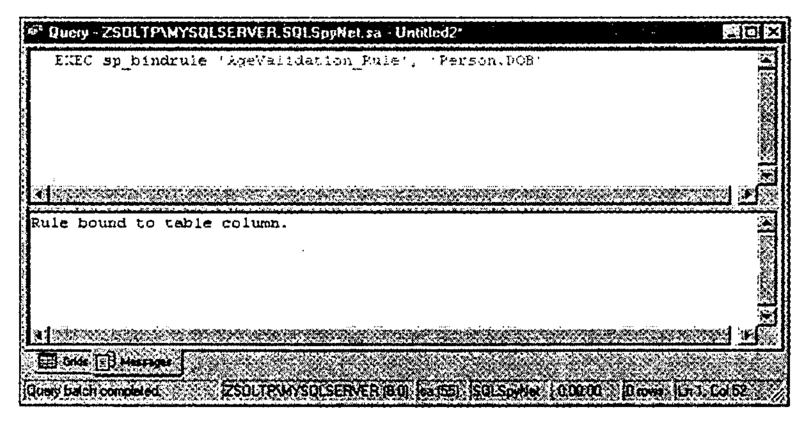


Рис. 7.1. SQL Server 2000 привязал наше первое правило к столбцу DOB таблицы Person

Теперь при попытке внесения в таблицу `Person` информации о человеке, еще не достигшем 16-летнего возраста, SQL Server 2000 будет возвращать сообщение об ошибке (рис. 7.2).

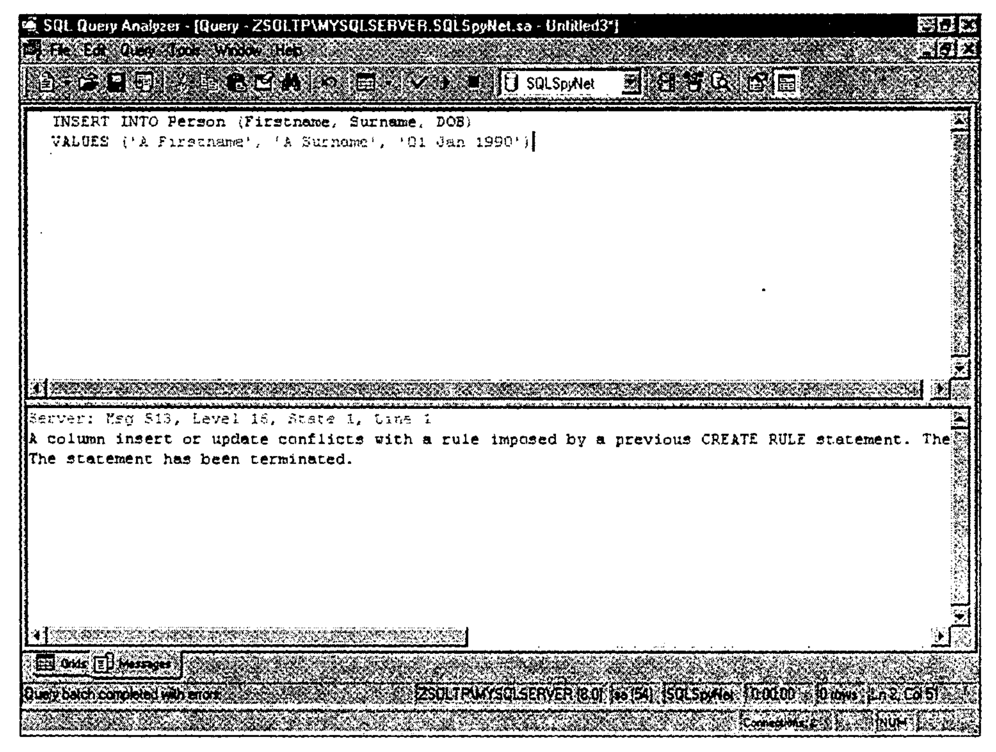


Рис. 7.2. SQL Server 2000 отменяет операцию `INSERT INTO` ввиду нарушения правила, определенного для столбца `DOB` таблицы `Person`

Для того чтобы в этом убедиться, введите и выполните код, приведенный в листинге 7.3.

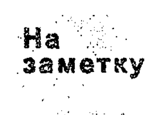
Листинг 7.3. Проверка работоспособности правила `AgeValidation_Rule`

Код для запуска

```
INSERT INTO Person (Firstname, Surname, DOB)
VALUES ('A Firstname', 'A Surname', '01 Jan 1990')
```

Не правда ли, создание и привязка правила не вызвали у вас особых затруднений?

Обратите внимание на текст сообщения об ошибке, возвращенного SQL Server 2000 в результате выполнения кода листинга 7.2. Для среднестатистического пользователя это, мягко говоря, выглядит не очень-то дружелюбно. В главе 8, “Защита данных с помощью транзакций, блокировок и механизма обработки ошибок”, рассматривается способ захвата возвращаемых SQL Server 2000 ошибок; кроме того, будет изменен текст отображаемого пользователю сообщения.



Чтобы удостовериться в работоспособности правила AgeValidation_Rule, попробуйте внести в таблицу Person записи с различными значениями поля DOB. Например, попробуйте внести в эту таблицу запись о человеке, возраст которого больше 16 лет или сведения о возрасте которого отсутствуют.

При внесении в таблицу Person записи о человеке, сведения о дате рождения которого отсутствуют (т.е. равны NULL), созданное правило активизировано не будет. Почему? Вспомните материал главы 1, "SQLSpyNet: от идеи до воплощения в виде базы данных SQL Server 2000", в которой впервые была затронута проблема, связанная с присутствием значений NULL в базе данных. Поскольку значение NULL не может быть больше, меньше или равно любому числовому значению, созданное правило не сможет корректно справиться с этой ситуацией.

Подводя итог этого раздела, следует отметить, что наш "багаж" пополнился еще одним инструментом, с помощью которого можно обеспечить целостность хранящейся в базе данных информации. Замечательно, не правда ли?

В следующем разделе рассмотрим использование стандартных значений.

Определение стандартных значений

Еще одним методом обеспечения целостности хранящейся в базе данных информации является определение стандартных значений. При создании таблиц базы данных SQLSpyNet мы уже назначили несколько стандартных значений для различных столбцов этих таблиц. Так, стандартным значением столбца DatePlanAttempted таблицы Activity является значение, возвращаемое функцией GETDATE().

Каким же образом определение стандартных значений может обеспечить целостность хранящейся в базе данных информации? Ответ очень прост: если пользователь по каким-либо причинам не знает значение, которое необходимо ввести в определенный столбец таблицы, SQL Server 2000 вместо неопределенного значения NULL (что, как вы помните, является далеко не лучшим решением) подставит в него некоторый стандартный "заполнитель". Если со временем недостающая информация обнаружится, то стандартное значение будет легко заменено значением, отражающим действительность.

Кроме возможности уменьшить количество присутствующих в базе данных неопределенных значений NULL, определяемые пользователем стандартные значения просто незаменимы в той ситуации, когда одно и то же стандартное значение должно находиться сразу в нескольких столбцах различных таблиц. Например, таким столбцом мог бы быть телефонный номер в таблицах Spy, BadGuy и Address. Разумеется, требуемое стандартное значение можно определить на этапе создания каждой из этих таблиц, что, однако, подразумевает неоднократное переписывание программного кода.

Вот в этом-то и заключается еще одно преимущество пользовательского стандартного значения: его достаточно создать всего один раз, а при назначении столбцам таблиц использовать механизм ссылок.

Однако наиболее значительным преимуществом является возможность изменять стандартное значение в одном-единственном месте, что, тем не менее, влечет за собой изменения во всех использующих это значение столбцах базы данных.

Создание стандартного значения

Чтобы чувствовать себя более уверенно, воспользуемся оператором CREATE DEFAULT и создадим стандартные значения для столбцов SpyNumber и Alias таблицы Spy, а также для столбца KnownAs таблицы BadGuy.

Поскольку все эти столбцы позволяют вводить значение NULL, пользователю не требуется обязательно заполнять их при вводе информации в базу данных. Смею вас заверить, что подобное "безобразие" непременно завершится сотнями или даже тысячами нежелательных значений NULL. А это плохо, плохо и еще раз плохо! Создание стандартных значений позволит указать SQL Server 2000 на необходимость автоматической подстановки определенного значения в случае его пропуска пользователем. В нашем случае этим стандартным значением будет unknown.

Введите код листинга 7.4 в окно ввода кода программы Query Analyzer.

Листинг 7.4. Создание стандартного значения

Код для запуска

```
CREATE DEFAULT Spy_BadGuy_Default AS 'unknown'
```

Аналогично правилу, работоспособность стандартного значения не может быть проверена до тех пор, пока оно не будет привязано к конкретному столбцу (или столбцам) базы данных. Для привязки стандартного значения следует воспользоваться встроенной хранимой процедурой SQL Server 2000 sp_bindefault. Ее синтаксис напоминает синтаксис хранимой процедуры sp_bindrule.

```
sp_bindefault 'стандартное_значение', 'имя_объекта',
↳'флаг_последующего_использования'
```

Единственное отличие хранимой процедуры sp_bindrule от sp_bindefault — разные первые аргументы, принимаемые этими процедурами. Как вы уже могли догадаться, аргумент *стандартное_значение* представляет собой имя стандартного значения, которое необходимо привязать к столбцу таблицы. В рассматриваемом случае это стандартное значение Spy_BadGuy_Default.

Введите код листинга 7.5 в окно ввода кода программы Query Analyzer.

Листинг 7.5. Привязка стандартного значения к столбцам таблиц Spy и BadGuy

Код для запуска

```
EXEC sp_bindefault 'Spy_BadGuy_Default',
'Spy.SpyNumber'
GO
EXEC sp_bindefault 'Spy_BadGuy_Default', 'Spy.Alias'
GO
EXEC sp_bindefault 'Spy_BadGuy_Default',
'BadGuy.KnownAs'
GO
```

Анализ

Обратите внимание, что в приведенном коде стандартное значение Spy_BadGuy_Default привязывается к трем столбцам таблиц Spy и BadGuy, что требует трех отдельных вызовов встроенной хранимой процедуры sp_bindefault. Между вызовами хранимой процедуры

Анализ

располагаются операторы GO, предназначенные просто для наглядной демонстрации выполнения кода Transact-SQL. В результате выполнения кода SQL Server 2000 возвратит сообщение, аналогичное показанному на рис. 7.3.

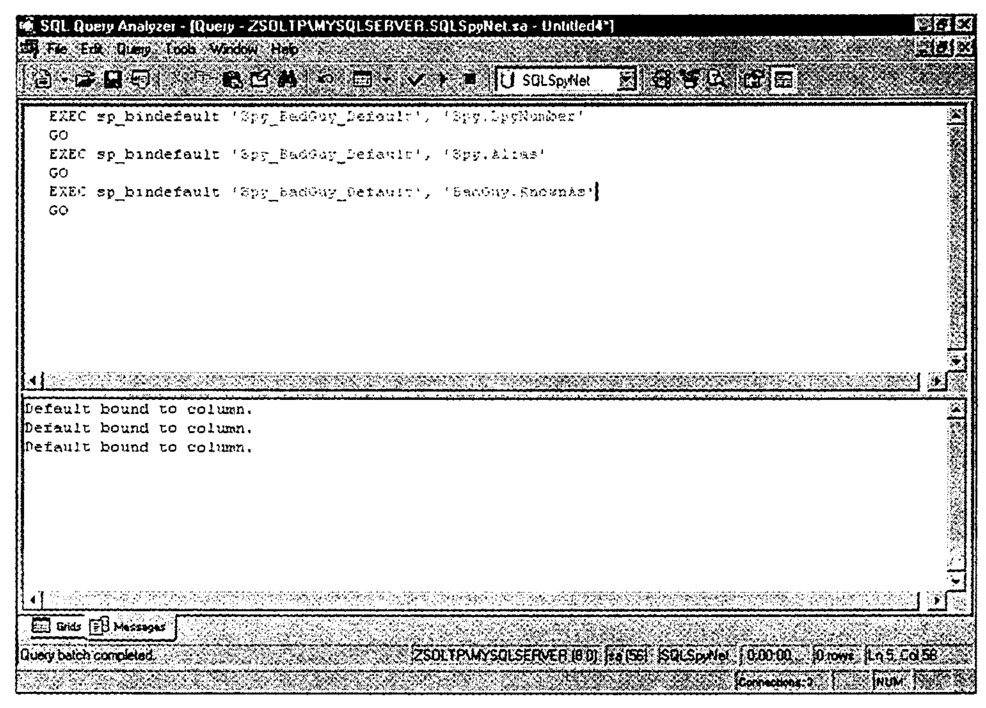


Рис. 7.3. SQL Server 2000 привязывает стандартное значение к трем столбцам таблиц Spy и BadGuy

Проверка работоспособности стандартного значения

После создания и привязки стандартного значения следует убедиться в его работоспособности. Для этого внесем новые записи в таблицы Spy и BadGuy, опустив при этом значения столбцов SpyNumber, Alias и KnownAs. Если все пройдет должным образом, то SQL Server 2000 автоматически подставит вместо пропущенных значений строку unknown.

Введите код листинга 7.6 в окно ввода кода программы Query Analyzer.

Листинг 7.6. Проверка работоспособности стандартного значения Spy_BadGuy_Default

Код для запуска

```
1: INSERT INTO Spy (
1a: PersonID, DateCommencedWork, AnnualSalary, IsActive)
2: VALUES (2, '23 May 1969', 152000.00, 1)
3: SELECT * FROM Spy WHERE PersonID = 2
```

Результат выполнения приведенного выше кода показан на рис. 7.4.

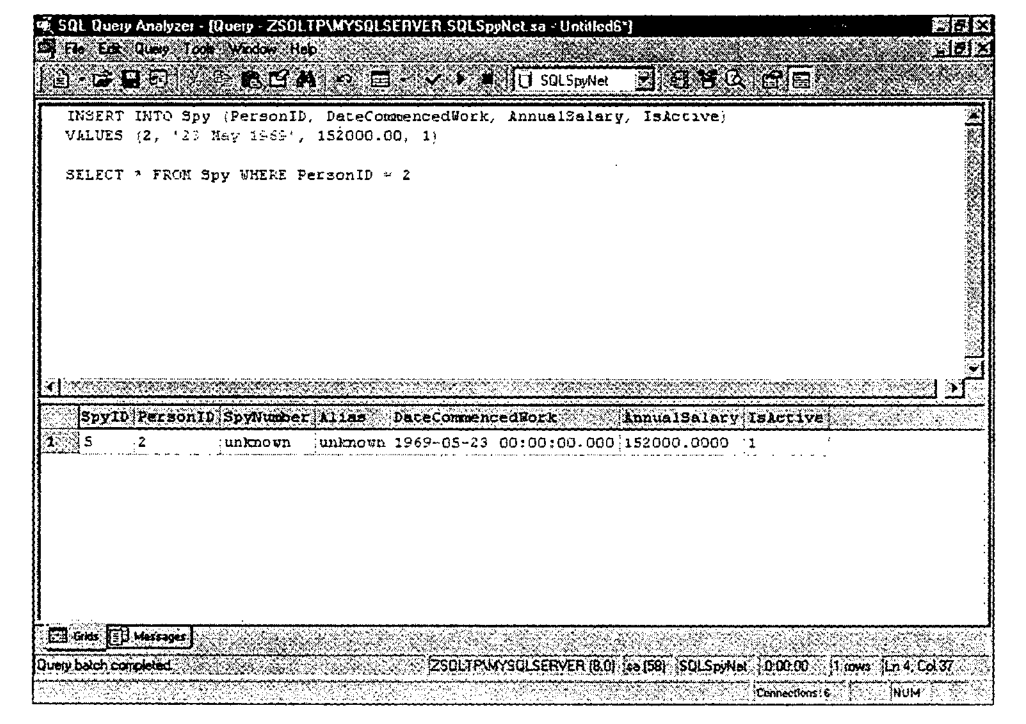


Рис. 7.4. SQL Server 2000 автоматически подставляет стандартные значения в столбцы `SpyNumber` и `Alias` таблицы `Spy`

Обратите внимание, что вместо пропущенных значений столбцов `SpyNumber` и `Alias` таблицы `Spy` SQL Server 2000 автоматически подставил стандартное значение `unknown`.

На заметку Для того чтобы быть уверенным в работоспособности стандартного значения в любых ситуациях, создайте еще несколько сценариев внесения новых записей в таблицу `Spy`. Например, попробуйте внести строку, указав значение столбца `Alias` и опустив значение столбца `SpyNumber` или наоборот. Наконец, попробуйте внести в таблицу `Spy` новую запись, указав значения всех ее столбцов.

Аналогичный тест следует провести по отношению к таблице `BadGuy`. Уверен, у вас вполне достаточно знаний, чтобы справиться с этой не очень трудной задачей.

На заметку Поскольку вы все еще “немножко новичок” в области Transact-SQL, ниже приведен исходный код, извлекающий из таблицы `Spy` все записи, содержащие значение `'unknown'` в полях `SpyNumber` и `Alias`. При необходимости эти значения можно изменить с помощью рассмотренного в главе 4, “Обработка данных с помощью кода Transact-SQL”, оператора `UPDATE`.

```
SELECT * FROM Spy
WHERE SpyNumber = 'unknown'
OR Alias = 'unknown'
```

В результате выполнения приведенного выше кода SQL Server 2000 возвратит все записи таблицы `Spy`, содержащие `'unknown'` в столбцах `SpyNumber` и `Alias`. При желании вы можете легко подставить в них действительные значения.

Поддержка единообразного представления информации с помощью пользовательских типов данных

Определяемые пользователем типы данных базируются на типах данных, встроенных в SQL Server 2000. Применяя пользовательские типы данных при создании таблиц, можно добиться большой гибкости, связанной прежде всего с обеспечением непротиворечивости хранящейся в базе данных информации.

Представим себе, что в структуру таблиц Spy, BadGuy и Address входит столбец с именем PhoneNumber, который был объявлен как столбец типа VARCHAR(14), и разрешает ввод значений NULL. Определяя пользовательский тип данных для столбца PhoneNumber, следует выбрать в качестве базового встроенный в SQL Server 2000 тип VARCHAR и разрешить поддержку этим типом данных значений NULL.

Теперь рассмотрим ситуацию, когда над проектом работают одновременно два разработчика, причем первый абсолютно убежден в том, что столбец PhoneNumber имеет тип данных VARCHAR(10), а не VARCHAR(14). Подобная несогласованность грозит вылиться в противоречивость информации во всей базе данных.

Для того чтобы быть уверенным, что несколько различных столбцов в базе данных имеют одинаковый тип, размер и способность хранить (или не хранить) в себе значения NULL, следует создать для них единый определяемый пользователем тип данных.

Применив пользовательский тип данных ко всем требующим этого столбцам, мы достигнем цели — обеспечим непротиворечивость информации во всей базе данных. Неплохо, правда?

Следует отметить, что изменение определяемого пользователем типа данных немедленно отразится на всех столбцах этого типа, что позволит без особых усилий реализовать что-то наподобие централизованного управления проектом.

Однако самое интересное заключается в том, что, помимо обеспечения единообразного представления информации в базе данных, определяемый пользователем тип данных позволяет привязать стандартное значение!

В том случае, если при вводе информации в базу данных пользователь по каким-либо причинам не укажет значение этого столбца, вместо него будет автоматически подставлено соответствующее стандартное значение. Более того, к определяемому пользователем типу данных можно привязать правило, которое будет обеспечивать корректный ввод информации в столбец этого типа.

Таким образом, объединив различные инструменты обеспечения целостности хранящейся в базе данных информации, можно добиться высокой надежности разрабатываемого приложения. Итак, не теряя времени, создадим первый пользовательский тип данных.

Проверка правильности ввода телефонных номеров

Несмотря на то что создание пользовательского типа данных не составляет особого труда, придется выполнить несколько дополнительных действий, чтобы обеспечить его корректное использование.

Так что же необходимо сделать? Прежде всего создадим тип данных с именем Person_PhoneNo. Этот определяемый пользователем тип будет базироваться на встроенном типе данных SQL Server 2000 VARCHAR(14). Далее создадим стандартное значение для типа Person_PhoneNo. В частности, стандартное значение должно гарантировать, что столбец с типом данных Person_PhoneNo не будет содержать значений NULL.

Наконец, создадим и привяжем правило, не позволяющее вводить в столбец типа Person_PhoneNo значение длиной менее 10 символов. Последним шагом будет использование оператора ALTER TABLE для изменения структуры таблицы Person, а именно внесения в нее нового столбца PersonNo, имеющего тип данных Person_PhoneNo.

На заметку

Наверняка вы уже отметили отличие используемого выше подхода от рассмотренной ранее "классической" схемы, когда стандартные значения и правила привязывались непосредственно к столбцам таблиц. Следует отметить, что привязка правил и стандартных значений к типу данных намного эффективнее привязки к столбцам таблиц. В этом случае мы получаем так называемую "инкапсуляцию" всей бизнес-логики в типе данных, что позволяет эффективно реализовать ее в необходимых местах приложения.

Для того чтобы создать определяемый пользователем тип данных, следует воспользоваться встроенной хранимой процедурой SQL Server 2000 sp_addtype, которая имеет следующий синтаксис:

```
1: sp_addtype
1a: 'тип', 'системный_тип_данных',
1b: 'допускает_ли_null', 'имя_владельца'
```

Анализ

Код в строке 1 говорит сам за себя: это имя хранимой процедуры, предназначенной для создания пользовательского типа данных.

С помощью параметров, расположенных в строке 1a, можно определить имя пользовательского типа данных и базовый системный тип, на основе которого этот тип данных будет создан. В рассматриваемом примере новый тип Person_PhoneNo создается на основе базового типа VARCHAR(14).

Первый параметр в строке 1b позволяет указать, допускает ли пользовательский тип данных ввод значений NULL. В случае пропуска этого параметра SQL Server 2000 будет использовать определенное для базы данных значение по умолчанию. Второй параметр используется для назначения владельца нового типа данных. По умолчанию владельцем созданного типа данных является текущий пользователь.

Процесс создания определяемого пользователем типа данных можно разделить на несколько этапов.

1. Введите код из листинга 7.7.

Листинг 7.7. Создание определяемого пользователем типа данных

Код для запуска

```
1:sp_addtype 'Person_PhoneNo', 'VARCHAR(14)', 'NOT NULL'
```

В результате экран будет выглядеть, как на рис. 7.5.

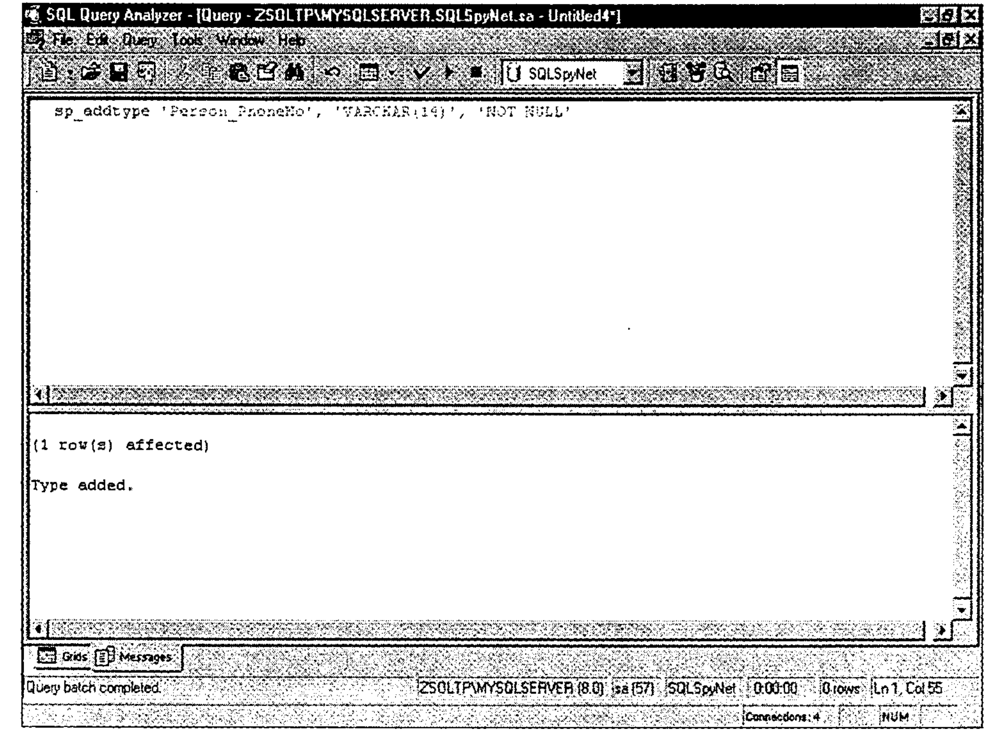


Рис. 7.5. SQL Server 2000 создает новый тип данных

2. Теперь необходимо создать стандартное значение и привязать его к новому типу данных. Для этого введите код листинга 7.8 в окно ввода кода программы Query Analyzer.

Листинг 7.8. Создание стандартного значения для нового типа данных

Код для запуска

```
CREATE DEFAULT PhoneNo_Default AS 'Not Available'
```

3. Привяжите стандартное значение к новому типу данных с помощью кода, приведенного в листинге 7.9.

Листинг 7.9. Привязка стандартного значения к новому типу данных

Код для запуска

```
EXEC sp_bindefault 'PhoneNo_Default',
'Person_PhoneNo'
```

Единственное отличие этого кода от кода Transact-SQL, который использовался для привязки стандартного значения в предыдущем разделе, — последний

аргумент хранимой процедуры sp_bindefault. В приведенном выше коде это тип данных, ранее же мы использовали имя столбца таблицы.

4. И наконец, мы должны создать правило и привязать его к новому типу данных. Введите код листинга 7.10 в окно ввода кода программы Query Analyzer.

Листинг 7.10. Создание правила для нового типа данных

Код для запуска

```
1: CREATE RULE PhoneNoLength_Rule
2: AS
3: LEN(@PhoneNo) > 9
```

5. Для того чтобы привязать правило к новому типу данных, следует еще раз воспользоваться процедурой sp_bindrule.

```
1: EXEC sp_bindrule 'PhoneNoLength_Rule',
1a:'Person_PhoneNo'
```

И опять единственным отличием этого кода от кода Transact-SQL, который использовался для привязки правила в предыдущем разделе, является использование в качестве последнего аргумента хранимой процедуры sp_bindrule имени правила вместо имени столбца таблицы.

А теперь сделаем небольшую паузу. Прежде чем вносить изменение в таблицу Person, мы должны убедиться в успешном выполнении всех предыдущих шагов. Запустите Enterprise Manager и выберите из расположенного в левой части главного окна этой программы дерева объектов базу данных SQLSpyNet.

В соответствующей базе данных SQLSpyNet ветви объектов отыщите папку User Defined Data Types (Определяемые пользователем типы данных). Именно здесь должен находиться только что созданный новый тип данных Person_PhoneNo.



Если вы не обнаружили в списке пользовательских типов базы данных *SQLSpyNet* тип Person_PhoneNo, щелкните на папке *User Defined Data Types* правой кнопкой мыши и выполните команду *Refresh* (Обновить). Если вам и на этот раз не повезло, постарайтесь вспомнить имя базы данных, для которой вы создали новый тип!

Щелкните дважды на типе данных Person_PhoneNo. Появится диалоговое окно, изображенное на рис. 7.6.

Обратите внимание, что в раскрывающемся списке Rule (Правило) содержится определенное для нового типа данных правило. Аналогичное утверждение касается и раскрывающегося списка Default (Стандартное значение).



Если раскрывающийся список *Rule* или *Default* пуст, закройте диалоговое окно и обновите содержимое соответствующих базе данных *SQLSpyNet* папок *Rules* (Правила) и *Defaults* (Стандартные значения). После этого вновь откройте диалоговое окно свойств нового типа данных — уверен, что теперь все будет в полном порядке!

Итак, только что создан вполне корректный со всех точек зрения пользовательский тип данных. Он базируется на встроенном типе данных SQL Server 2000 VARCHAR(14) и имеет привязанные к нему стандартное значение и правило. Уверен, что теперь вас мучает всего лишь один вопрос: каким образом можно его использовать?

Рис. 7.6. SQL Server 2000 отображает диалоговое окно со свойствами пользовательского типа данных

Использование нового типа данных

Итак, наконец-то мы подошли к тому моменту, когда новый тип данных будет реализован в таблице `Person`. Для этого следует ввести код листинга 7.11 в окно ввода кода программы Query Analyzer.

Листинг 7.11. Изменение структуры таблицы *Person* с целью реализации в ней нового типа данных

Код для запуска

```
1: ALTER TABLE Person
2: ADD PhoneNo Person_PhoneNo
```

В результате выполнения этого кода SQL Server 2000 возвратит сообщение, подтверждающее успешное внесение изменений в таблицу `Person`.



Рассмотрим структуру оператора изменения таблицы Transact-SQL `ALTER TABLE`.

- Код, расположенный в строке 1, указывает SQL Server 2000 необходимость изменения структуры таблицы, а также имя изменяемой таблицы.

- В строке 2 определяется тип вносимых в таблицу изменений. В данном случае к структуре таблицы Person добавляется один новый столбец. После ключевого слова ADD, определяющего тип изменения, указывается имя и тип данных нового столбца таблицы. В рассматриваемом случае столбец носит имя PhoneNo, а его тип, как вы уже наверняка догадались, — это созданный ранее пользовательский тип данных Person_PhoneNo.

На заметку При добавлении нового столбца в таблицу мы не воспользовались ключевыми словами DEFAULT и CONSTRAINT, определяющими соответственно его значение по умолчанию и накладываемые ограничения. Спросите, почему? Да ведь мы уже задали все это в определяемом пользователем типе данных!

Для того чтобы убедиться в корректном внесении изменений в таблицу Person, запустите программу Enterprise Manager, отыщите в дереве объектов SQL Server 2000 базу данных SQLSpyNet и найдите соответствующую ей папку Tables (Таблицы). Дважды щелкните на таблице Person; появится диалоговое окно, изображенное на рис. 7.7.

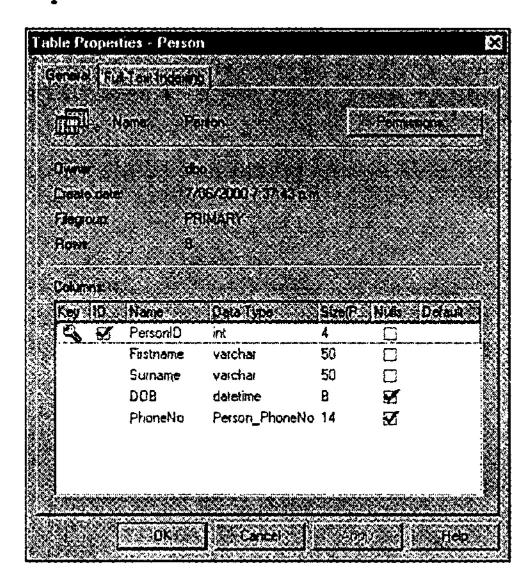


Рис. 7.7. SQL Server 2000 отображает диалоговое окно свойств таблицы Person, в структуре которой присутствует добавленный новый столбец



На заметку Возможно, вы обратили внимание на тот интересный факт, что в соответствии с отображаемой SQL Server 2000 информацией новый столбец таблицы Person позволяет вносить в него значения NULL, хотя на этапе определения типа данных Person_PhoneNo было указано прямо противоположное свойство. Дело в том, что в SQL Server 2000 к уже созданной таблице невозможно добавить новый столбец со свойством NOT NULL. Спросите, почему? Ну естественно, потому что вы добавляете абсолютно пустой с точки зрения находящейся в нем информации столбец! Таким образом, даже явно запретив внесение значений NULL на уровне типа данных, мы не сможем повлиять на текущее положение вещей.

Диалоговое окно свойств таблицы Person отображает, в частности, и ее структуру, в которой появился новый столбец PhoneNo.

Что теперь? Разумеется, проверка работоспособности внесенных в таблицу изменений! Как всегда, я снабжу вас несколькими примерами тестов, что, однако же, не избавит вас от необходимости протестировать все более скрупулезно.

Введите код листинга 7.12 в окно ввода кода программы Query Analyzer.

Листинг 7.12. Проверка работоспособности столбца PhoneNo и определяемого пользователем типа данных Person_PhoneNo

Код для запуска

```
INSERT INTO Person (Firstname, Surname, DOB)
VALUES ('Bert', 'Renault', '10 Jul 1950')
SELECT * FROM Person WHERE Firstname = 'Bert'
```

Поскольку в приведенном коде намеренно было пропущено поле PhoneNo таблицы Person, SQL Server 2000 автоматически заполнил его определенным при создании типа стандартным значением, как показано на рис. 7.8.

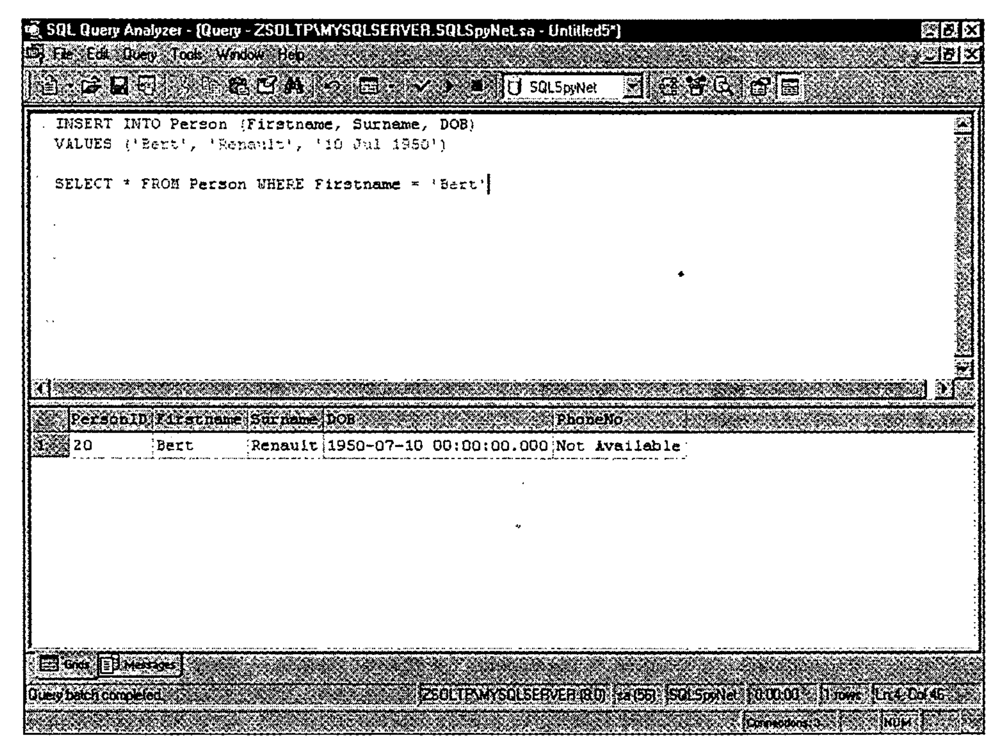


Рис. 7.8. SQL Server 2000 отображает результат определения стандартного значения для типа данных Person_PhoneNo

А теперь проверим работоспособность правила PhoneNoLength_Rule, которое, напомним, запрещает ввод в поле PhoneNo значения, в котором менее 10 символов.

Введите код листинга 7.13 в окно ввода кода программы Query Analyzer.

Листинг 7.13. Проверка работоспособности правила PhoneNoLength_Rule

Код для запуска

```
1: INSERT INTO Person (Firstname, Surname, DOB, PhoneNo)
2: VALUES ('The', 'Jokerette', '17 Nov 1962', '123')
```

SQL Server 2000 запретит выполнение оператора INSERT, поскольку значение поля PhoneNo имеет длину менее 10 символов.

Дополнительная проверка работоспособности правила PhoneNoLength_Rule полностью возлагается на ваши плечи. Вообще говоря, это не очень-то сложная задача: необходимо проверить возможность ввода в поле PhoneNo значения длиной более 9 символов, а также запрет выполнения оператора INSERT при попытке ввода в поле PhoneNo значения NULL. И наконец, аналогичные тесты следует выполнить для операции обновления данных UPDATE.

На этом заканчивается обсуждение определяемых пользователем типов данных, правил и стандартных значений. Как видите, эффективность и легкость использования этих инструментов обеспечения целостности хранящейся в базе данных информации поистине фантастическая.

Резюме

Рассмотренный в этой главе материал является "последним штрихом" в вопросе обеспечения целостности хранящейся в базе данных информации. Теперь для обеспечения непротиворечивости данных можно пользоваться четырьмя основными типами: целостностью на уровне сущностей, ссылок, доменов, а также определяемой пользователем. Только благодаря этому приложение SQLSpyNet уже можно рассматривать как достаточно серьезный проект!

Следует отметить, что, несмотря на внедрение нескольких отличных методов обеспечения достоверности информации, не существует универсального способа обезопасить приложение от ввода пользователями некорректных данных. Вернее, есть: строго-настрого запретить им изменять хранящуюся в базе данных информацию!

Таким образом, целостность данных — отличная вещь, и ее поддержка в равной степени лежит как на разработчике базы данных, так и на ее пользователях.

Следующие шаги

В следующей главе рассматриваются такие ключевые моменты использования СУРБД в многопользовательском режиме, как обработка ошибок, транзакции и блокировки. Итак, поехали!

ГЛАВА 8

Защита данных с помощью транзакций, блокировок и механизма обработки ошибок

В этой главе...

В этой главе рассматриваются способы обеспечения сохранности находящейся в базе данных информации. (Не от воров, а от пользователей, чье плохое обращение с базой данных может причинить не меньше вреда.)

При создании приложения разработчик неизбежно допускает некоторые предположения относительно использования своего продукта. Народная мудрость гласит: "Человек предполагает, а Бог располагает", однако в данном случае это единственный способ хоть как-нибудь уберечь приложение от "нашествия варваров" (т.е. конечных пользователей). Итак, следует всегда отдавать себе отчет в том, что конечная (хотя, может быть, и абсолютно непреднамеренная) цель всех пользователей — "сломать" ваше приложение. (Как бы там ни было, всегда найдется один "умник", который либо введет не то, что нужно, либо щелкнет не на той кнопке.)

Для того чтобы защитить приложение от подобных неприятностей, следует познакомиться с такими ключевыми концепциями использования СУРБД в многопользовательском режиме, как транзакции, блокировки и механизм обработки ошибок, в результате применения которых ваше приложение, несомненно, станет гораздо менее уязвимо со стороны всякого рода случайностей.

Итак, наша конечная цель определена: это программный код со стопроцентной гарантией отсутствия в нем каких-либо ошибок. Несмотря на всю сложность задачи, ее необходимо хотя бы попытаться выполнить.

Для начала рассмотрим транзакции и постараемся уяснить их основополагающую роль в механизме обеспечения непротиворечивости хранящейся в базе данных информации. Как это часто бывает, блокировки и транзакции идут, что называется, рука об руку. Мы детально рассмотрим само понятие блокировки данных и факторы, которые обязательно необходимо учитывать при ее реализации. И наконец, познакомимся с механизмом обработки ошибок, который, как вы уже наверняка догадались, приходится задействовать в самом последнем случае.

Обеспечение непротиворечивости данных с помощью транзакций

В предыдущих главах рассматривался способ изменения информации, хранящейся в базе данных SQLSpyNet. Как вы помните, для этого было необходимо написать и выполнить несколько операторов Transact-SQL произвольной сложности. Но как же быть в той ситуации, когда изменения в базу данных уже внесены и вдруг выясняется, что они могут привести к потере целостности данных или нарушению определенных бизнес-правил?

В наиболее простых случаях ситуацию можно исправить путем повторного ввода (восстановления) старой информации; к сожалению, это не спасет положения, если вносимые в базу данных изменения затронули сразу несколько связанных между собой таблиц. В этой ситуации восстановление информации представляет собой действительно нешуточную задачу. Что же делать? Ответ прост: научиться использовать транзакции.

Термин *Транзакция* позволяет сгруппировать и последовательно выполнить несколько операторов Transact-SQL. Используя транзакции, можно организовать код в виде выполняющихся один за другим логических блоков. "Ну и что же тут особенного? — спросите вы. — Ведь в предыдущих главах этой книги мы уже научились группировать операторы Transact-SQL с помощью хранимых процедур, триггеров и функций. До сих пор блок операторов `BEGIN...END` прекрасно справлялся с этой задачей".

Все это действительно так, однако использование транзакций позволяет добиться намного большей гибкости при изменении информации в базе данных. С помощью транзакций вы можете провести несколько изменений информации подряд, а затем, в зависимости от объективных условий, принять или отменить внесенные изменения. Это несколько напоминает использование параметра Track Changes (Исправления) в программе Microsoft Word, который позволяет подтвердить или отменить изменения, внесенные в текущий документ.

Звучит великолепно, но какое отношение это имеет к приложению SQLSpyNet? Начнем с самого простого: транзакции позволяют нам, а также пользователям нашей базы данных проводить обновления хранящейся в ней информации. При возникновении ошибки во время обновления можно отменить его и проинформировать пользователя о случившемся. Таким образом, дополнительная функциональность, предоставляемая транзакциями, помогает избежать появления в базе данных противоречивой информации, а также позволяет обрабатывать ошибки, что приводит к снижению вероятности сбоя приложения в результате какой-либо внештатной ситуации.

Более подробно механизм обработки ошибок рассматривается далее в главе.

Для того чтобы вы могли лучше представить себе, что же такое транзакция, я расскажу небольшую, но очень убедительную историю. Итак, слушайте...

Перенесемся на несколько лет в прошлое, в те времена, когда вы еще не занимали высокий пост руководителя программы компании Spy Net Limited, а были всего лишь одним из помощников в аппарате старого руководства.

Сидя за своим столом (или столиком), вы получили срочный приказ, скрепленный печатью самого Президента! С неистовой скоростью вы бросились набирать программный код, результатом выполнения которого должна была стать ракетная атака против одного из коварнейших врагов Spy Net доктора Зло, который захватил маленький островок в Тихом океане и готовит свой очередной план захвата всей планеты.

С самодовольной ухмылкой (еще бы: отлично выполненное задание, к тому же в рекордные сроки!) вы откинулись в кресле и приготовились насладиться чашечкой любимого кофе. Однако, взглянув на монитор, вы отметили нечто странное в поведении ракетных установок: угол, на который они разворачивались, не совсем соответствовал маленькому островку в Тихом океане, более того, это был определенно не Тихий океан! В гробовом молчании вы проводили глазами несколько дюжин ракет, летящих в неизвестном (пока) направлении. Похоже, что ваш код историки будут упоминать впоследствии как причину третьей мировой войны (если после нее, конечно, останется хотя бы один историк!). Вряд ли дипломаты замнут такой инцидент — ну с чего бы вдруг Президент объявил войну всему миру?

Вспомнив введенный код, вы (о ужас!) обнаружили грубейшую ошибку в одной из его главных частей. То, что вы ввели, выглядело как *Казнить нельзя, помиловать*, тогда как следовало ввести *Казнить, нельзя помиловать*.

Тем временем дома рушились, автомобили опрокидывались, в воздух взлетали части разорванных тел из-за мощнейших взрывов, потрясавших в этот момент города всего мира, — и все это было результатом одной-единственной ошибки в программном коде.

Вы взяли пистолет, медленно взвели курок и приготовились достойно уйти, как вдруг вас осенило... В мгновение ока вы подбежали к столу и что есть сил закричали: "ОТМЕНА!".

Глядя на монитор, вы не могли нарадоваться своей сообразительности: здания восстанавливались из пепла, автомобили как ни в чем не бывало мчались по шоссе, люди, живые и здоровые, шли по своим делам, даже ракеты, изменив направление, возвращались на базу — словом, мир стал таким, каким он был до выполнения того злополучного кода.

Пользуясь терминологией, принятой в сфере разработки баз данных, вы только что отменили свою первую транзакцию. Несмотря на то что мой рассказ несколько "экстремален", он прекрасно демонстрирует выполнение транзакции и то, что называется ее отменой (ROLLBACK).

А теперь рассмотрим механизм, который позволил нам предотвратить конец света.

Итак, транзакция началась с выполнения кода, инициирующего запуск ракет. Ввод кода послужил началу так называемой *неявной* транзакции. Более подробно различные типа транзакций (неявные, явные и автоматически фиксируемые) будут рассмотрены далее в главе.

После выполнения кода мы явно указали на необходимость его отмены для того, чтобы получить еще один шанс спасти мир.



Один из моих приятелей намекнул на возможность сравнения транзакции с известным фильмом *День Сурка* (Groundhog Day). Его герой в исполнении великолепного Билла Мюррея (Bill Murray) снова и снова просыпается в одном и том же дне. Независимо от того, как герой проводит его и какие поступки совершает, каждый день заканчивается по-разному; но проснувшись, герой наверняка знает, что все вернется на свои места. Аналогичным этому явлением можно считать открытие одной и той же транзакции и совершение различных действий над хранящейся в базе данных информацией с их последующей отменой.

Итак, в рассмотренной "транзакции" были сгруппированы такие события, как позиционирование установок для пуска ракет, пуск ракет, разрушение зданий и т.д. К счастью, она удовлетворяла всем правилам, налагаемым на транзакции, что в конечном итоге позволило сохранить мир на планете.

ACID-тест: требования, предъявляемые к транзакциям

Транзакции должны удовлетворять некоторым требованиям. Как следует из названия этого раздела, требования, предъявляемые к транзакциям, известны также как ACID-тест, где A — это *atomicity* (*атомарность*), C — *consistency* (*согласованность*), I — *isolation* (*изолированность*), D — *durability* (*устойчивость*).

Термин Свойство *атомарности* обозначает, что транзакция не может быть выполнена частично. Произойдет "либо все, либо ничего".

Термин *Согласованность* обозначает, что выполнение транзакции не должно привести к непредвиденным изменениям в хранящейся в базе данных информации. В частности, это подразумевает выполнение всех условий реляционной базы данных (например, целостность на уровне первичных ключей).

Термин Свойство *изолированности* обозначает, что транзакция должна быть автономной и не должна воздействовать на другие транзакции или зависеть от них. Дело в том, что транзакции могут быть вложены друг в друга. Свойство изолированности подразумевает, что если одна транзакция изменяет данные, то следующая может получить доступ к ним либо перед, либо после их изменений со стороны первой транзакции. Таким образом, вторая транзакция не должна "видеть" данные в процессе их изменения первой транзакцией.

Термин *Устойчивость* обозначает, что после завершения транзакции внесенные в хранящуюся в базе данных информацию изменения останутся постоянными. Другими словами, все останется как есть, даже если что-то случится с системой!

При проектировании транзакции следует учитывать перечисленные правила и стараться писать согласованный с ними программный код. Несмотря на то что SQL Server 2000 по умолчанию поддерживает некоторые из ACID-правил, именно разработчик приложения должен отвечать за окончательное проектирование транзакции.

Подведем некоторый итог. Как видите, транзакция — это всего лишь "оболочка" для группы выполняемых операторов Transact-SQL. Используя некоторые полезные свойства этой оболочки, можно зафиксировать или, наоборот, отменить внесенные в хранящуюся в базе данных информацию изменения, вернув таким образом систему в ее первоначальное состояние.

Выбор типа транзакции

Если вы все еще не уверены в целесообразности использования транзакций, отбросьте всякие сомнения и поспешите выяснить, каким же способом может быть создана транзакция. При разработке приложения SQLSpyNet мы будем использовать только явные (к тому же достаточно компактные) транзакции; тем не менее SQL Server 2000 позволяет сконфигурировать механизм выполнения транзакций несколькими способами.

В SQL Server 2000 существует три основных варианта определения транзакций.

- Неявные транзакции. Данный вариант должен быть указан вручную. Неявные транзакции можно задействовать либо автоматически при подключении пользователя к базе данных, либо явно с помощью команды SET IMPLICIT_TRANSACTIONS ON. И в том и в другом случае будет открыта неявная транзакция, которая впоследствии должна быть зафиксирована (COMMIT) или отменена (ROLLBACK).

На заметку Максимального эффекта от использования неявных транзакций можно добиться при подключении клиентского приложения к экземпляру SQL Server 2000.

Неявную транзакцию следует открыть при установке соединения, чтобы затем, после внесения пользователем изменений в базу данных (UPDATE, INSERT или DELETE), зафиксировать либо, наоборот, отменить их.

- Автоматически фиксируемые транзакции. Это стандартный вариант использования механизма транзакций в SQL Server 2000. В зависимости от ситуации, SQL Server 2000 автоматически зафиксирует (COMMIT) или отменит (ROLLBACK) результат выполнения оператора Transact-SQL. По умолчанию попытка закрыть транзакцию совершается сразу же после выполнения оператора Transact-SQL. В этом смысле принцип работы SQL Server 2000 схож с принципом работы базы данных Microsoft Access. Следует отметить, что, даже если вы используете неявные или явные транзакции, после их завершения SQL Server 2000 снова возвратится к автоматической фиксации транзакций.
- Явные транзакции. В этом случае явно указывается начало и конец каждой транзакции. Завершив вносить изменения в базу данных, вы можете их либо зафиксировать (COMMIT), либо отменить (ROLLBACK).

С этого момента мы будем явно отмечать начало и конец каждой транзакции, так как это позволяет более гибко контролировать механизм их выполнения. Тем не менее следует помнить, что после завершения транзакции SQL Server 2000 возвратится к принципу их автоматической фиксации.

Итак, мы обсудили все, кроме самого главного — способа объявления транзакций. Поскольку в дальнейшем при разработке приложения SQLSpyNet будут использоваться только явные транзакции, вопрос определения начала и конца транзакции приобретает всю большую актуальность.

Оператор явного объявления транзакции состоит всего из двух частей.

- BEGIN TRANSACTION или BEGIN TRAN. Инициирует начало транзакции. Это первая часть оператора определения транзакции.

Далее представлена его вторая часть.

- COMMIT TRAN или COMMIT TRANSACTION. Один из вариантов второй части оператора определения транзакции. Фиксирует изменения, внесенные в базу данных в результате выполнения входящих в транзакцию операторов Transact-SQL.
- ROLLBACK TRAN и ROLLBACK TRANSACTION. Это второй вариант. Отменяет изменения, внесенные в базу данных в результате выполнения входящих в транзакцию операторов Transact-SQL, возвращая тем самым систему в состояние, в котором она находилась до выполнения транзакции. Как видите, вторая часть оператора определения транзакции подразумевает необходимость фиксации (COMMIT) или отмены (ROLLBACK) изменений в базе данных.

Внутри оператора BEGIN TRABSACTION...COMMIT/ROLLBACK TRANSACTION может находиться как один, так и несколько операторов Transact-SQL. При этом, однако, следует отметить наличие нескольких крайне неприятных ловушек, которые будут рассмотрены в ближайших разделах этой главы.

Экскурс

Использование двухфазной фиксации изменений

Несмотря на кажущуюся простоту применения транзакций в пределах одной базы данных, расширение масштаба до двух и более баз данных кардинальным образом меняет ситуацию.

В случае неудачи мы производим отмену изменений, внесенных в базу данных в результате выполнения операторов транзакции. Все это выглядит просто по отношению к одной базе данных, но что же происходит при использовании транзакции, вносящей изменения в несколько различных баз данных? В качестве примера рассмотрим старую аналогию с переводом денег из одной страны в другую.

Итак, вы идете в банк и переводите сумму в 1 000 долларов на один из своих заграничных счетов. Разумеется, два различных банка не могут использовать одну и ту же базу данных, а кроме того, непременно должны обеспечивать эффективность проведения подобных операций.

Пока деньги, снятые с вашего счета, не поступят на заграничный счет, транзакция не может считаться завершенной. В противном случае не исключена ситуация, в которой деньги, снятые со счета, так и не придут в место своего назначения.

Двухфазная фиксация изменений позволяет проводить эффективное управление транзакциями, использующими две различные базы данных.

Теперь, когда вы имеете хотя бы некоторое представление о двухфазной фиксации изменений, рассмотрим более подробно сам принцип ее работы.

Первая транзакция открывается в момент снятия денег с вашего "отечественного" счета, в то время как вторая — в момент поступления денег на заграничный счет. Безусловно, этого далеко не достаточно для управления всем процессом синхронизации транзакций. Здесь на помощь приходит *диспетчер ресурсов* (*resource manager*) — программа, которая обеспечивает все вовлеченные в транзакцию ресурсы информацией о состоянии транзакции.

Диспетчер ресурсов в SQL Server 2000 носит название *координатора распределенных транзакций Microsoft* (*Microsoft Distributed Transaction Coordinator — MS DTC*). Координатор запрашивает каждый из участвующих в транзакции ресурсов о готовности к фиксации изменений. Эта операция известна также как *подготовка* транзакции.

Как только все ресурсы сообщат о готовности к фиксации изменений (другими словами, подтвердят отсутствие ошибок), координатор распределенных транзакций позволит завершиться всем открытым на данный момент транзакциям уже без своего непосредственного участия. С другой стороны, если координатор получит хотя бы одно сообщение о невозможности фиксации изменений, он разошлет широковещательное сообщение всем ресурсам, предупреждающее их о необходимости отмены транзакции. Такой метод фиксации призван обеспечить одно из присущих транзакции свойств — атомарность.

Все это напоминает метод "негативного поощрения", который используется некоторыми менеджерами при управлении персоналом. Пока все идет хорошо,

менеджер остается незаметным. Однако стоит вам совершить одну ничтожную ошибку... О, этот "разговор" вы запомните на всю оставшуюся жизнь!

Экскурс

Подводя итог, следует отметить, что координатор распределенных транзакций — надежный залог вашей уверенности в том, что ни вы, ни тем более банк не останетесь "в пролете".

Создание транзакции для приложения SQLSpyNet

Не правда ли, разговор о транзакциях вызвал у вас непреодолимое желание создать что-нибудь свое? (Признаться, именно этого я и добивался.)

В приведенном ниже примере рассматриваются базовые аспекты, связанные с объявлением и использованием транзакции. Для этого в таблицу AddressType вносятся некоторые заведомо некорректные изменения: вместо значения *Physical House Address* в столбец Description помещается значение *Physical Horse Address*. После этого благодаря использованию транзакции эти изменения отменяются.

На самом деле этот пример вряд ли сможет послужить основой для создания более-менее полезной транзакции, реально использующейся в разрабатываемом нами приложении. Нечто действительно стоящее будет создано только после изучения механизма обработки ошибок, а пока наша задача состоит лишь в поверхностном знакомстве с транзакцией и способом ее выполнения.

Итак, запустите программу Query Analyzer и введите код листинга 8.1.

Листинг 8.1. Проверка выполнения транзакции, вносящей изменения в таблицу AddressType

Код для запуска

```
1:  SELECT AddressTypeID, Description FROM AddressType
1a:  WHERE AddressTypeID = 4
2:  BEGIN TRANSACTION
3:  UPDATE AddressType SET Description =
           ' Physical Horse Address'
3a:  WHERE AddressTypeID = 4
4:  SELECT AddressTypeID, Description FROM AddressType
4a:  WHERE AddressTypeID = 4
5:  ROLLBACK TRANSACTION
6:  SELECT AddressTypeID, Description FROM AddressType
6a:  WHERE AddressTypeID = 4
```

Проанализируем приведенный код Transact-SQL.

Анализ

- Расположенный в строке 1 оператор SELECT извлекает информацию из таблицы AddressType до выполнения транзакции.
- Расположенный в строке 2 оператор BEGIN TRANSACTION указывает SQL Server 2000 на необходимость открытия новой транзакции.
- В строке 3 выполняется изменение хранящейся в таблице AddressType информации.
- Расположенный в строке 4 оператор SELECT подтверждает внесение изменений в таблицу AddressType. Как показано на рис. 8.1, в поле Description этой таблицы было помещено заведомо некорректное значение **Physical Horse Address**.
- В строке 5 указывается необходимость отмены внесенных в таб-



лицу AddressType изменений. Для этого используется оператор ROLLBACK, который отменяет изменения, внесенные в ходе текущей транзакции. Помимо этого, оператор ROLLBACK закрывает и завершает текущую транзакцию.

- Расположенный в строке 6 оператор SELECT предназначен для демонстрации результата отмена транзакции.

На рис. 8.1 представлен результат выполнения кода, наглядно иллюстрирующего использование транзакции.

SQL Query Analyzer - [Query - ZSOLTP\MYSQLSERVER.SQLSpyNet.sa - Untitled1*]

File Edit Query Tools Window Help

```
SELECT AddressTypeID, Description FROM AddressType
WHERE AddressTypeID = 4

BEGIN TRANSACTION
        UPDATE AddressType SET Description = 'Physical Horse Address'
        WHERE AddressTypeID = 4

        SELECT AddressTypeID, Description FROM AddressType
        WHERE AddressTypeID = 4
ROLLBACK TRANSACTION

SELECT AddressTypeID, Description FROM AddressType
WHERE AddressTypeID = 4
```

	AddressTypeID	Description
1	4	Postal Business

	AddressTypeID	Description
1	4	Physical Horse Address

	AddressTypeID	Description
1	4	Postal Business

Grids Messages

Query batch completed. ZSOLTP\MYSQLSERVER (8.0) sa (51) SQLSpyNet 0:00:00 3 rows Ln 11, Col 1

Connections: 1 NUM

Рис. 8.1. Наглядная демонстрация использования операторов BEGIN *и* ROLLBACK TRANSACTION

Итак, мы получили то, что хотели, — достаточно простую транзакцию, которая изменяет хранящуюся в таблице AddressType информацию, наглядно демонстрируя при этом состояние данных на всех этапах выполнения. Кроме всего прочего, созданная транзакция удовлетворяет набору ACID-правил.



Использование транзакции с ее последующей отменой (ROLLBACK) может быть полезным при тестировании операторов Transact-SQL, которые теоретически могут нанести вред хранящейся в базе данных информации. Представим себе, что в базу данных SQLSpyNet необходимо внести изменения, связанные с повышением заработной платы всех агентов на 10%.

Однако, прежде чем зафиксировать эти изменения, следует убедиться в том, что новые расходы не "прорвут дыру" в бюджете компании на следующий год.

Используя транзакцию, можно внести изменения в базу данных, проверить бюджетное ограничение, а затем при необходимости отменить "раздачу



слонов". Конечно, вам предстоит в весьма деликатной форме постараться убедить сотрудников, что компания переживает не лучшие времена и необходимо еще немножко подождать.

Уверен, что, поразмыслив, вы найдете еще немало интересных примеров использования транзакций для проверки определенных условий.

Следует отметить, что существует целый ряд способов повышения эффективности использования транзакций; они рассматриваются в следующем разделе.

Несколько способов повышения эффективности использования транзакций

При разработке транзакций следует учитывать приведенные ниже соображения.

- Старайтесь создавать как можно более короткие транзакции. Почему? Дело в том, что SQL Server 2000 использует довольно много ресурсов для того, чтобы убедиться в соответствии транзакции ACID-правилам. Поэтому вполне естественно, что слишком длинные транзакции могут привести к избыточному расходованию системных ресурсов. Хотя это замечание и не касается в полной мере систем, рассчитанных на нескольких пользователей, оно крайне важно по отношению к действительно многопользовательским системам, производительность которых может существенно зависеть от правильного выбора стратегии создания транзакций.
- Разрабатывайте эффективные стратегии блокировки во избежание ситуаций, в которых пользователь может извлечь из базы данных информацию, внесенную в нее еще не завершившейся транзакцией. Более подробно стратегии блокировки рассматриваются в следующем разделе.
- Исключите возможность ввода информации в базу данных во время выполнения транзакции. Конечно, человек намного медленнее реагирует на событие, нежели компьютер, а, как упоминалось ранее, выполнение транзакции требует достаточно большого количества ресурсов системы. Именно поэтому очень важно, чтобы пользователи ввели всю необходимую информацию еще до начала выполнения транзакции.
- Исключите возможность открытия транзакции во время просмотра хранящейся в базе данных информации. Прежде всего это поможет избежать внештатных ситуаций, касающихся блокировки данных. К тому же транзакции предназначены для обеспечения защиты информации во время проведения операций обновления, а не извлечения данных.
- Старайтесь уменьшать объем хранящейся в базе данных информации, которая каким-либо образом затрагивается при выполнении транзакции. Например, проводя обновление информации всего лишь в одной строке таблицы, удостоверьтесь в том, что вы блокировали только эту строку, а не таблицу целиком. Как правило, подобная осмотрительность с лихвой оправдывает себя, так как во многом способствует уменьшению числа ошибок при блокировке данных.

Этот список можно дополнить еще несколькими важными правилами, однако вы уже получили достаточно информации для того, чтобы приступить к изучению второго средства обеспечения целостности хранящейся в базе данных информации — механизма блокировки данных.

Использование блокировок для поддержки целостности хранящейся в базе данных информации

Блокировка (*locking*), подобно транзакции, представляет собой одну из ключевых концепций в сфере разработки реляционных баз данных. В ходе создания приложения SQLSpyNet мы не будем явно разрабатывать стратегию блокировки, тем не менее нельзя отрицать того, что любое приложение базы данных должно легко поддаваться масштабированию без кардинального пересмотра его структуры. Именно поэтому изучение механизма блокировки информации очень важно в свете последующей адаптации приложения SQLSpyNet к использованию на корпоративном уровне.

Один из наиболее наглядных примеров, позволяющих лучше понять механизм блокировки, — управление счетами в банке. Представьте себе, что вы женаты и на вашем семейном счете, не предусматривающем превышения кредита, лежит 400 долларов. Вы идете в банк и пытаетесь снять со счета 300 долларов. В это же время ваша жена идет в другой банк и пытается снять со счета 200 долларов. Кассир в вашем банке обрабатывает запрос и, видя 400 долларов на текущем счете, выдает вам необходимую сумму в 300 долларов. Еще до окончания операции снятия денег кассир в другом банке аналогичным образом обрабатывает запрос и выдает вашей жене 200 долларов. Когда обе транзакции наконец-то закроются, кассиры с удивлением обнаружат превышение кредита вашего семейного счета на 100 долларов.

Как же следует поступить в такой ситуации? Когда первый кассир инициировал операцию по снятию денег со счета, этот счет должен был быть автоматически блокирован. Блокировка счета не позволила бы второму кассиру начать операцию по снятию денег до завершения первой операции. Более подробно все аспекты блокировки данных рассматриваются в ближайших разделах этой главы, а пока следует запомнить, что процесс разработки транзакций требует более пристального внимания, чем могло показаться вначале.

Блокировка не позволяет считывать изменяемую информацию; кроме того, она способна предотвратить одновременное обновление одних и тех же данных разными пользователями. Таким образом, блокировка, как и транзакция, призвана обеспечить непротиворечивость хранящейся в базе данных информации. Несмотря на то что нам вряд ли удастся полностью избежать ввода некорректной информации со стороны пользователей, мы, по крайней мере, сможем должным образом обработать ее в нашем приложении.

Автоматическая блокировка данных со стороны SQL Server 2000

SQL Server 2000 автоматически управляет большинством происходящих в базе данных процессов без вмешательства со стороны администратора или разработчика приложения. Для того чтобы просмотреть список ресурсов, блокированных пользователем, следует обратиться к программе Enterprise Manager.

Итак, рассмотрим способ определения местонахождения диспетчера блокировок в SQL Server 2000.

1. Запустите Enterprise Manager и отыщите в дереве объектов папку Management (Управление).

2. В папке Management находится папка Current Activity (Текущая активность пользователей), которая содержит информацию о текущей активности пользователей, существующих блокировках информации и выполняющихся процессах.

3. Для того чтобы просмотреть информацию о существующих в данный момент блокировках, откройте папку Locks/Object (Блокировки/Объект). На рис. 8.2 представлено окно программы Enterprise Manager, в левой части которого находится список блокированных объектов SQL Server 2000.

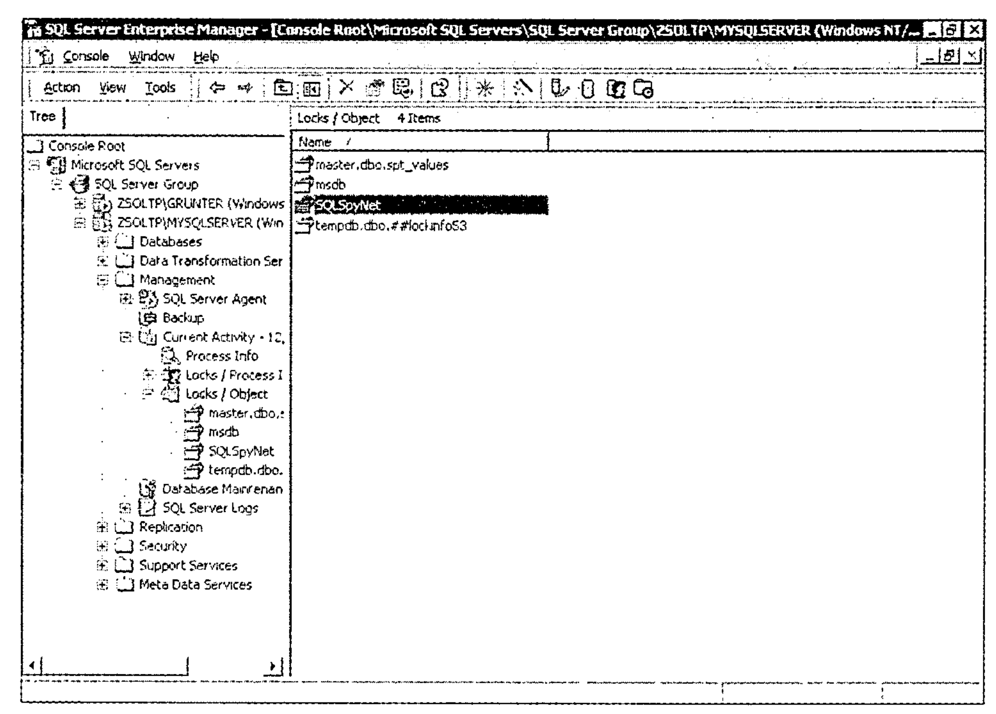


Рис. 8.2. SQL Server 2000 отображает список блокированных объектов, в число которых входит и база данных SQLSpyNet

4. Для того чтобы узнать, какой пользователь инициировал блокировку, щелкните дважды на пиктограмме базы данных SQLSpyNet, расположенной в папке Locks/Object.

На заметку

При просмотре блокировок объектов базы данных инициировавшие их пользователи представляются специальным идентификатором ProcessID (или SPID). Это значение динамически присваивается SQL Server 2000 и обозначает номер текущего пользовательского процесса.

Видите, даже SQL Server 2000 использует где-то там внутри себя первичные ключи!

Как же на основании значения идентификатора SPID определить имя пользователя? Информация обо всех процессах, запущенных в рамках SQL Server 2000, хранится во внутренней таблице под названием *sysprocesses*. Кроме всего прочего, эта таблица хранит номер процесса SPID и имя пользователя (LoginName), инициировавшего данный процесс. Как видите, определение имени пользователя на основе номера процесса — задача действительно тривиальная.

Как правило, встроенный механизм автоматического управления блокировкой SQL Server 2000 корректно назначает блокировки ресурсов в большинстве возможных ситуаций. На основании информации о запросе и базовой логической структуре ресурса SQL Server 2000 принимает решение о степени гранулированности блокировки.

Несмотря на то что автоматическое назначение блокировок не так уж и плохо (и даже совсем наоборот), очень часто возникает необходимость взять процесс блокировки ресурсов под свой контроль.

Экскурс

Как заставить SQL Server 2000 блокировать дверь (простите, таблицу!) по нашему желанию

Для того чтобы это сделать, следует воспользоваться так называемой директивой блокировки. *Директива блокировки* (*locking hint*) позволяет явно указать тип блокировки, который должен использовать SQL Server 2000, в результате чего оптимизатор запросов применит эту блокировку к выполняемому оператору SELECT. Кроме директивы блокировки, существуют также директивы таблиц, представлений и объединений.

Например, в приведенном ниже коде мы используем блокировку для обеспечения взаимоисключающего доступа к таблице Person, которая будет применена SQL Server 2000 при попытке извлечения информации из этой таблицы.

```
1: SELECT Firstname FROM Person WITH (TABLOCKX)
```

В результате выполнения этого кода SQL Server 2000 будет использовать блокировку таблицы Person при выполнении оператора SELECT, что, кстати, не могло быть сделано автоматически.

Ах да, совсем забыл! Никогда не используйте блокировку для обеспечения взаимоисключающего доступа при извлечении из нее информации!

Как правило, блокировка назначается для определенного объема информации и длится также вполне определенное время. Представьте себе, что вам понадобилось взять дело Джеймса Бонда из общедоступного шкафа для хранения документов. Имея дело в своем распоряжении, вы можете его читать и изменять, однако при этом, естественно, никто больше не может получить к нему доступ. Таким образом, в данный момент вы имеете эксклюзивное (взаимоисключающее) право на использование материалов по Джеймсу Бонду! Но не обольщайтесь: как только вы вернете дело в шкаф для хранения документов (в терминах реляционных баз данных это означает снятие блокировки), любой желающий сможет снова взять его для чтения или внесения изменений.

Приведенная выше аналогия в точности отражает принцип использования блокировки в SQL Server 2000. После того как пользователь получил в свое распоряжение определенный ресурс, этот ресурс блокируется и никто больше не может получить к нему доступ до тех пор, пока блокировка не будет снята.

Существует несколько степеней блокировки, начиная от низкоуровневой блокировки столбцов и заканчивая высокоуровневой блокировкой таблиц. Как правило, использование различных типов блокировки сказывается на производительности приложения базы данных.

На заметку

Как и транзакции, блокировки требуют использования значительных ресурсов сервера. Бездарно разработанная стратегия блокировки может легко привести попросту к остановке приложения базы данных. Случайно блокировав и не освободив определенный ресурс, можно лишить пользователей возможности выполнять повседневные задачи.

Разработка стратегии блокировки

Создавая приложение с перспективой его дальнейшего расширения, следует уделить особое внимание разработке эффективной стратегии блокировки. При увеличении числа пользователей такого приложения вы можете быть уверены в том, что это не скажется критическим образом на его производительности. Поэтому при разработке широкомасштабного приложения всегда старайтесь заглянуть в будущее с тем, чтобы уже сейчас попытаться избежать некоторых возможных проблем.

Проблемы, связанные с одновременным доступом к данным

Если ваша система поддерживает режим работы более чем с одним пользователем, она становится местом потенциального возникновения *проблем, связанных с одновременным доступом к данным*. Эти проблемы можно разделить на четыре основные категории.

- Проблемы, связанные с одновременным обновлением информации. Два пользователя одновременно вносят изменения в запись одного и того же тайного агента в таблице Spy. В результате изменения, внесенные первым пользователем, теряются.
- Недействительный результат чтения (проблема несохраненных данных). Предположим, некий пользователь вносит изменения в запись тайного агента. Второй пользователь считывает еще не зафиксированную информацию, в то время как первый обнаруживает ошибку и отменяет транзакцию. Таким образом, второй пользователь не знает о том, что данные не были сохранены, и принимает решение на основе неверной информации.
- Невозможность повторного чтения (проблема противоречивого анализа информации). Первый пользователь дважды считывает информацию о тайном агенте, однако в промежутке между считываниями второй пользователь полностью изменяет эту информацию.
- Проблема "фантомного" считывания информации. Первый пользователь считывает и обновляет информацию о тайном агенте. Затем второй пользователь пытается отразить внесенные в данную информацию изменения в приложении, однако при этом он неожиданно обнаруживает еще одно изменение, внесенное в эту же информацию с "третьей" стороны.

Итак, как же нам избежать подобных внештатных ситуаций? Наиболее эффективное решение — блокировка информации, не позволяющая другому пользователю считать или обновить ее до тех пор, пока она не будет "освобождена" первым пользователем.

Довольно очевидно, не правда ли? К сожалению, далеко не всегда удается блокировать доступ к ресурсу для определенного пользователя. Кроме всего прочего, кто-то же должен продолжать работать!

Выбор типа блокировки для решения проблем, связанных с одновременным доступом к данным

Что же делать? Не отчаиваться и выбрать для решения каждой конкретной проблемы один из приведенных ниже типов блокировки.

- Проблемы, связанные с одновременным обновлением информации. Не позволять пользователю вносить изменения в информацию до завершения внесе-

ния изменений первым пользователем. С одной стороны, пользователь сохраняет возможность считывания данных, а с другой — до поры до времени не может их изменить.

- Недействительный результат чтения. В этом случае необходимо запретить пользователям считывать данные до завершения их обновления. Чтобы реализовать подобную стратегию, следует применить блокировку для обеспечения взаимоисключающего доступа к ресурсам (вспомните пример с делом Джеймса Бонда).
- Невозможность повторного чтения. Как и для проблемы, связанной с недействительным результатом чтения, следует запретить пользователям считывать данные до завершения операции по их обновлению.
- Проблема "фантомного" считывания информации. В данном случае следует блокировать доступ к информации для всех пользователей на то время, пока первый и третий пользователь вносят в нее изменения.

Таким образом, разнообразию проблем, связанных с работой в многопользовательской среде, мы можем противопоставить разнообразие стратегий блокировки, которые могут быть применены по отношению к хранящейся в базе данных информации. Какие же существуют уровни блокировки?

Выбор уровня блокировки

Рассматривая уровни блокировки, следует использовать понятие *грануляции* данных.

Термин *Грануляция* (*granularity*) определяет степень детализации блокировки. Так, блокирована может быть как вся таблица (высокий уровень грануляции данных), так и одна-единственная запись (низкий уровень грануляции данных).

Выбирая уровень блокировки, следует учитывать некоторые немаловажные моменты. Низкий уровень грануляции данных требует использования большого числа системных ресурсов, обеспечивая при этом высокую гарантию непротиворечивости хранящейся в базе данных информации. Использование же высокоуровневой грануляции приводит к более "скромной" загрузке системы, однако это повышает риск несогласованности информации и может привести к блокированию некоторых важных ресурсов, что негативно скажется на возможности пользователей продолжать работу в нормальном режиме. Здесь следует отметить еще раз, что, как правило, SQL Server 2000 в состоянии решить за вас большинство проблем, связанных с блокировкой информации.

SQL Server 2000 использует динамический принцип блокировки информации при выборе наиболее эффективной (а значит, и сберегающей ресурсы) стратегии блокировки транзакций. В зависимости от запроса и лежащей в основе логической структуры, SQL Server 2000 выбирает наиболее оптимальный для выполнения определенной задачи уровень блокировки.

Что же делать в той ситуации, когда вы хотите явным образом указать SQL Server 2000 необходимый уровень блокировки? Как правило, вам вряд ли понадобится делать это для приложения, рассчитанного на работу всего лишь нескольких пользователей, однако иногда становится совершенно очевидно, что блокировка некоторых ресурсов снимается не так быстро, как этого хотелось бы. Именно в таких случаях необходимо явно указать наиболее оптимальный, с вашей точки зрения, уровень блокировки.

SQL Server 2000 позволяет блокировать перечисленные ниже ресурсы (начиная с низкоуровневых и заканчивая высокоуровневыми).

- Идентификатор записи. Позволяет блокировать отдельную запись в таблице. Наиболее низкий из всех доступных уровней грануляции данных.
- Ключ. Позволяет блокировать запись внутри индекса.
- Страница. Позволяет блокировать страницу таблицы или индекса. Имеет более низкий уровень, чем блокировка таблицы, так как таблица, как правило, состоит из нескольких страниц.
- Экстент. Позволяет блокировать группу из восьми расположенных подряд страниц.
- Таблица. Позволяет блокировать таблицу, включая все ее данные и индексы.
- База данных. Как видите, SQL Server 2000 позволяет блокировать даже целую базу данных! Разумеется, это наиболее высокий из всех доступных уровней блокировки информации.

Помимо выбора уровня блокировки, SQL Server 2000 поддерживает шесть основных *режимов* блокировки ресурсов.

- Разделяемая блокировка (shared lock). Наиболее "либеральный" режим, позволяющий всем пользователям считывать информацию блокированного ресурса. Вместе с тем данный режим запрещает какое-либо изменение информации до снятия блокировки. Блокировка снимается после выполнения всех операций считывания данных.
- Блокировка обновления (update lock). Данный режим позволяет снизить вероятность возникновения взаимных блокировок. Одновременно только одна транзакция может установить блокировку обновления ресурса. При попытке изменения этого ресурса (например, удаления информации с помощью оператора DELETE) блокировка обновления автоматически переключается в режим монопольного использования, в противном случае — в режим разделяемой блокировки.
- Монопольная блокировка (exclusive lock). Используется для "изоляции" ресурса, при которой всем остальным пользователям запрещается как считывание, так и обновление информации. Один из наиболее высокоуровневых режимов блокировки в SQL Server 2000.
- Целевая блокировка (intent lock). Используется для блокировки ресурса на уровне страниц в режиме разделяемой или монопольной блокировки. Установка целевой монопольной блокировки ресурса фактически означает установку монопольной блокировки таблицы на уровне страниц и запрет блокировки этого ресурса со стороны других транзакций. Для чего это нужно? Дело в том, что внутреннему механизму SQL Server 2000 намного легче проверять блокировку ресурса (что делается в случае запроса блокировки со стороны других транзакций) на уровне таблицы, нежели каждую ее запись или страницу в отдельности.
- Блокировка на уровне логической структуры (schema lock). Существует два типа такой блокировки: разделяемый и монопольный. (Учитывая сказанное выше, вы уже наверняка догадались о выделении двух основных режимов блокировок: разделяемого и монопольного.) Монопольная блокировка на уровне логической структуры используется, например, при добавлении столбца к таблице, в то время как разделяемая — при компиляции запросов. Это позволяет другим транзакциям выполняться, однако запрещает вносить изменения в логическую структуру базы данных.

- Блокировка вследствие обновления (копирования) большого объема данных (bulk update lock). Как следует из названия, этот режим используется при копировании большого объема информации в таблицу базы данных. При установке блокировки этого типа все остальные транзакции (не выполняющие копирования большого объема данных) не могут выполнять операции чтения или обновления информации в таблице.

Итак, в вашем распоряжении весьма неплохой арсенал средств, используемых SQL Server 2000 для управления механизмом транзакций и блокировок. К сожалению, слишком много хорошего не бывает — существует одна проблема, предотвратить которую без вашей помощи не сможет даже SQL Server 2000. Она называется взаимная блокировка и ее во что бы то ни стало следует избегать.

Экскурс

Как избежать взаимной блокировки

Воистину, взаимная блокировка — это самая большая головная боль администратора базы данных. Взаимная блокировка не является прерогативой SQL Server 2000, она встречается также во многих других Windows-приложениях.

Так что же такое взаимная блокировка? *Взаимная блокировка* (*deadlock* или *deadly embrace*) возникает, когда пользователь заблокировал определенный ресурс и требует заблокировать еще один ресурс. К несчастью, этот ресурс уже блокирован другим пользователем, который также требует заблокировать еще один ресурс (догадайтесь, какой!) — тот, что был блокирован первым пользователем.

Таким образом, первый пользователь не может получить блокировку еще одного ресурса и поэтому не освобождает уже заблокированный ресурс, но и второй пользователь не может получить блокировку ресурса первого пользователя и также не освобождает свой ресурс. Весело, ничего не скажешь!

Для того чтобы нагляднее представить всю сложность взаимной блокировки, сравните ее с ситуацией, в которой два маленьких ребенка "сражаются" за игрушки. У одного из них есть грузовик, у другого — самолет, и каждый из них не сдастся до тех пор, пока не заполучит в свое распоряжение игрушку другого (естественно, сохранив при этом у себя свою). Итак, теперь вы понимаете какого масштаба проблема возникает в базе данных при появлении взаимной блокировки!

К счастью, внутренний механизм SQL Server 2000 использует специальный алгоритм обнаружения взаимных блокировок. Установив ее возникновение, SQL Server 2000 принудительно завершает транзакцию одного из пользователей (как правило, путем ее отмены), давая тем самым завершиться транзакции другого пользователя.

Несмотря на то что внутренний механизм SQL Server 2000 позволяет в большинстве случаев решить проблему взаимной блокировки, будьте особенно бдительны при разработке стратегии блокировки ресурсов в приложении, рассчитанном на многопользовательский режим работы.

Некоторые соображения, касающиеся стратегии блокировки

Как и в случае с транзакциями, разработка стратегии блокировки требует повышенного внимания. Использование только одного типа блокировки (с высокоуровне-

вой грануляцией данных) может быть приемлемо для приложения, рассчитанного на нескольких пользователей, однако при попытке расширить его до масштабов корпоративного приложения производительность системы часто оказывается ниже оптимальной. Поэтому при разработке стратегии блокировки данных старайтесь придерживаться перечисленных ниже базовых правил.

- Следуйте основным принципам разработки стратегии использования транзакций, которая имеет очень много общего со стратегией блокировки.
- Испытайте свое приложение в "утяжеленном режиме" (так называемый стресс-тест). "Утяжеленный режим" подразумевает работу приложения в моменты максимальной загрузки с учетом максимального числа подключенных пользователей.
- Сохраните для пользователей возможность аварийно завершать слишком долго выполняющиеся запросы. Например, поиск информации, возвращающий в результате 2 млн записей, может занять достаточно долгое время, так что пользователь должен иметь возможность его отменить.
- Следует запретить пользователям вводить данные во время запроса. Дело в том, что, пока выполняется запрос, SQL Server 2000 блокирует определенные ресурсы базы данных, в скорейшем разблокировании которых могут быть заинтересованы остальные пользователи.

Несмотря на возможность явной блокировки запросов и объектов базы данных, полностью доверимся в этом непростом вопросе диспетчеру блокировок SQL Server 2000. В будущем, возможно, мы пересмотрим свое отношение к этому вопросу, а пока давайте оставим все как есть.

Механизм обработки ошибок SQL Server 2000

Иногда вещи ломаются. К сожалению, исключение не составляет и разрабатываемое приложение: насколько бы искусным программистом вы не являлись, всегда найдется пользователь, которому взбредет в голову каким-либо совершенно невообразимым способом довести его до возникновения внештатной ситуации.

Если все будет развиваться по худшему сценарию, ошибка "ускользнет" незамеченной и проявится нарушением согласованности информации в базе данных вследствие успешного выполнения не совсем корректной операции обновления, удаления или вставки данных. Не стоит исключать вариант полного "краха" приложения, после чего вам, вероятно, не удастся избежать яростного негодования пользователей. Стоп! Так дело не пойдет!

Что же можно сделать для того, чтобы максимальным образом уменьшить вред, наносимый приложению внештатными ситуациями? Ниже приведен список довольно эффективных средств обработки ошибок, предлагаемых SQL Server 2000.

- Определение типа внештатной ситуации.
- Выбор действия, предпринимаемого в ответ на появление внештатной ситуации.
- Восстановление или отмена внесенных в базу данных изменений.
- Информирование пользователя о внештатной ситуации.

С помощью SQL Server 2000 можно создавать собственные сообщения о возникновении ошибок для более гибкого информирования пользователей о сбоях в работе приложения. Но для начала давайте рассмотрим сам способ обработки внештатной ситуации.

"Составные части" ошибки в SQL Server 2000

В одном из следующих разделов этой главы будет рассмотрена реализация обработки ошибки с помощью хранимой процедуры, а пока попробуем внести новую запись в таблицу BadGuy без наличия соответствующей ей записи в таблице Person.

Как вы, надеюсь, помните, правило целостности информации в базе данных гласит, что дочерняя запись не может существовать без соответствующей ей родительской записи. Таким образом, выполнение кода листинга 8.2 приведет к тому, что SQL Server 2000 запретит внесение информации в таблицу BadGuy ввиду отсутствия соответствующей ей родительской записи в таблице Person.

Листинг 8.2. Попытка внесения новой записи в таблицу BadGuy без соответствующей ей родительской записи в таблице Person

```
INSERT INTO BadGuy (PersonID, KnownAs, IsActive)
VALUES (10000, 'A real bad dude', 1)
```

Обратите внимание на сообщение, которое отобразит SQL Server 2000 в результате выполнения приведенного выше кода (рис. 8.3).

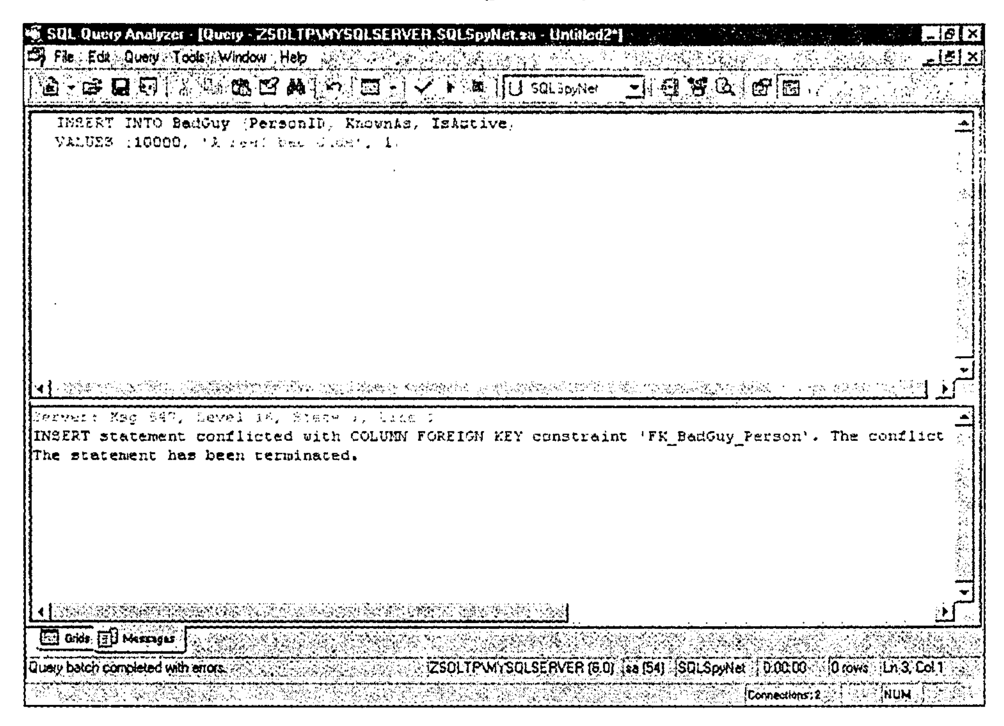


Рис. 8.3. SQL Server 2000 запрещает внесение информации в таблицу BadGuy и отображает системное сообщение об ошибке

Не удивительно, что большинство пользователей толком так и не поймут, что же произошло. Все, чего можно добиться с помощью такого сообщения, — это смутить и окончательно расстроить их. Для того чтобы приложение "не издевалось" над пользователями, следует реализовать собственный механизм обработки ошибок.

SQL Server 2000, наподобие Visual Basic и других современных языков программирования, имеет достаточно "продвинутый" объект для представления ошибок. Он содержит несколько атрибутов, которые можно использовать для получения дополнительной информации об ошибке.

- Номер ошибки. Это число, которое представляет собой внутренний номер ошибки и является результатом выполнения функции `@@ERROR`. Каждая ошибка имеет свой уникальный номер.
- Описание ошибки или строка с сообщением об ошибке. Информация, которую SQL Server 2000 возвращает пользователю в результате обнаружения ошибки. SQL Server 2000 может автоматически подставить в эту строку некоторые конкретизирующие ситуацию значения, например `Update on` *имя_таблицы* `failed`. Каждый раз при возврате этого сообщения SQL Server 2000 подставляет вместо атрибута *имя_таблицы* реальное имя таблицы, при обновлении данных которой возникла ошибка. Каждая ошибка имеет свое уникальное описание.
- Серьезность ошибки. Позволяет определить степень серьезности ошибки. Низкие значения (2 и меньше) соответствуют несущественным информативным ошибкам или предупреждениям, в то время как более высокие — ошибкам, которые мешают выполнению той или иной операции над базой данных.
- Код состояния. Используется в том случае, когда данная ошибка возникает во многих местах исходного кода SQL Server 2000. Уникальный код состояния присваивается сообщению об ошибке каждый раз при возникновении внештатной ситуации, что позволяет средству Microsoft SQL Server Engineer диагностировать возникшую проблему.
- Имя процедуры. Имеет значение только в том случае, когда ошибка возникла внутри хранимой процедуры.
- Номер строки. Представляет собой номер строки хранимой процедуры, выполнение которой привело к появлению ошибки.

SQL Server 2000 хранит все сообщения об ошибках в базе данных `master`, в специальной системной таблице под названием *sysmessages*.

Все интерфейсы (ODBC, ADO и т.д.), взаимодействующие с SQL Server 2000, обладают возможностью генерации отчетов об ошибках на основе перечисленных атрибутов (особенно на основе номеров и описаний ошибок). К сожалению, информацию, заключенную в более специализированных атрибутах, все эти интерфейсы трактуют по-разному.

Итак, мы рассмотрели способ внутреннего представления ошибок в SQL Server 2000. Осталось немного — научиться их обрабатывать.

Обработка ошибки

Для того чтобы определить номер ошибки, следует воспользоваться встроенной функцией SQL Server 2000 `@@ERROR`. В результате работы эта функция возвращает целое число. Если возвращенное функцией `@@ERROR` значение равно нулю (0), это фактически соответствует отсутствию ошибки. Если же оно отлично от нуля, значит, что-то пошло не так и вам предстоит принять меры, адекватные возникшей внештатной ситуации.



При проверке значения функции @@ERROR следует принять во внимание один очень важный момент. Дело в том, что значение этой функции обновляется каждый раз при выполнении очередного оператора Transact-SQL. Таким образом, вам необходимо либо проверять значение функции @@ERROR сразу после выполнения оператора Transact-SQL, либо сохранить его в локальной переменной для последующего анализа.

Итак, как же воспользоваться значением функции @@ERROR? Для этого необходимо "привлечь" условный оператор IF, описанный в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet". Ниже приведен шаблон "классического" примера обработки ошибки в SQL Server 2000:

```
IF @@ERROR <> 0
     BEGIN
    --Выполнить некоторое действие
END
```

Первое, что следует сделать, — проверить, не равно ли значение функции @@ERROR нулю. Если оно отлично от нуля (верный признак ошибки), остается предпринять необходимое действие.

Представьте себе, что вам необходимо провести обновление информации в таблице. Организовав процесс обновления таблицы в виде транзакции и обнаружив ошибку, вы сможете отменить транзакцию и уведомить пользователя о возникшей внештатной ситуации. Таким образом, механизм обработки ошибок может сослужить добрую службу при разработке надежного и стабильно работающего приложения.

При желании механизм обработки ошибок SQL Server 2000 можно использовать для проверки наличия заданной ошибки. Например, вы проводите обновление информации в таблице Spy, но тайного агента со значением поля PersonID, запись о котором вы намереваетесь внести в базу данных, в таблице Person не существует. В этом случае необходимо проверить значение функции @@ERROR на равенство 547 — коду ошибки ограничения внешнего ключа. После этого вы можете возвратить пользователю собственное сообщение, предупреждающее его о том, что, прежде чем вносить информацию в таблицу Spy, ее необходимо внести в таблицу Person.

Как заставить ошибку работать на вас

Итак, вы зафиксировали ошибку, но что же делать дальше? Бежать от нее сломя голову! Ну а если серьезно, то при обнаружении ошибки вы можете выполнить несколько стандартных действий.

- Отменить выполняющееся задание. При обнаружении ошибки во время выполнения какого-либо задания вашим первоочередным действием должна быть отмена любых внесенных до этого момента изменений в базу данных. Это, по крайней мере, должно уберечь вас от нарушения непротиворечивости хранящейся в базе данных информации.
- Немедленно завершить или продолжить выполнение текущего задания. Обнаружив ошибку, вы можете либо немедленно прекратить выполнение текущего оператора Transact-SQL, либо позволить ему выполняться дальше. Если ошибка представляет собой всего лишь низкоуровневое предупреждение и не угрожает производительности базы данных, вполне возможен вариант продолжения нормального выполнения текущего задания; в противном случае будет лучше все немедленно прекратить.

- Возвратить пользователю сообщение, разъясняющее причину сбоя при выполнении приложения. Несмотря на то что стандартное сообщение об ошибке, возвращаемое SQL Server 2000, также несет в себе много полезной информации (например, для разработчика или администратора базы данных), будет лучше, если вы постараетесь перевести его для пользователя на "человеческий" язык.

Напоследок рассмотрим хранимую процедуру, описанную в главе 5, "Использование языка определения данных для просмотра и обновления информации", чтобы добавить в нее средство обработки ошибок.

Добавление средства обработки ошибок в хранимую процедуру

В главе 5 было описано создание двух хранимых процедур, использующих оператор `INSERT` для вставки данных в таблицу `Person` и в таблицы `Spy` или `BadGuy`.

Если оставить все как есть, то мы допускаем возможность частичного внесения информации в базу данных. Так, в результате успешного выполнения первого оператора `INSERT` новая информация заносится только в таблицу `Person`. Если выполнение второго оператора `INSERT` завершится неудачно, мы получим "осиротелую" запись в таблице `Person`, которой не соответствует запись в таблицах `Spy` или `BadGuy`. Хуже того, если пользователь снова запустит эту же хранимую процедуру, то в результате мы получим две одинаковые записи в таблице `Person`, одной из которых не соответствует никакая запись в таблицах `Spy` или `BadGuy`.

Ниже приведен список изменений, которые необходимо внести в код хранимой процедуры для обеспечения гарантии ее корректного выполнения. Следует отметить, что в качестве примера здесь приводится обновленный код хранимой процедуры `PersonBadGuyInsert`, а код второй хранимой процедуры — `PersonSpyInsert` — вам предлагается изменить самостоятельно.

Итак, запустите программу Query Analyzer и выберите SQLSpyNet в качестве активной базы данных, после чего введите и выполните код листинга 8.3.

Листинг 8.3. Добавление средства обработки ошибок в хранимую процедуру `PersonBadGuyInsert`

Код для запуска

```
ALTER PROCEDURE PersonBadGuyInsert
   @Firstname VARCHAR(50),
   @Surname VARCHAR (50),
   @DOB DATETIME = NULL,
   @KnownAs VARCHAR(25) = NULL,
   @IsActive BIT = 1
AS
   DECLARE @LocalError INT
   BEGIN TRANSACTION
     INSERT INTO Person (Firstname, Surname, DOB)
     VALUES (@Firstname, @Surname, @DOB)
    SET @LocalError = @@ERROR
    DECLARE @PersonID INT
    SET @PersonID = IDENT_CURRENT('Person')
    INSERT INTO BadGuy (PersonID, KnownAs, IsActive)
    VALUES (@PersonID, @KnownAs, @IsActive)
    SET @LocalError = @LocalError + @@ERROR
    IF @LocalError = 0
```

```
19:         BEGIN
20:             COMMIT TRANSACTION
21:             PRINT 'You have successfully added a new
                 ➥Person and their Bad guy details'
22:         END
23:     ELSE
24:         BEGIN
25:             IF @LocalError = 547
26:                 BEGIN
27:                     ROLLBACK TRANSACTION
28:                     PRINT 'Oops. You must add a Person
                         ➥before you add a Bad guy!'
29:                 END
30:             ELSE
31:                 BEGIN
32:                     ROLLBACK TRANSACTION
33:                     PRINT 'Oops an error occurred
                         ➥please try again!'
34:                 END
35:         END
```

Особого внимания в этом коде заслуживают описанные ниже строки.

Анализ

- Строка 8, в которой объявляется локальная переменная для хранения значения функции @@ERROR после выполнения каждого из операторов Transact-SQL.
- Строка 9, в которой открывается транзакция для обеспечения возможности отмены внесенных в базу данных изменений в случае возникновения ошибки.
- Строка 12, в которой значение функции @@ERROR присваивается локальной переменной @LocalError. При возникновения ошибки значение этой переменной будет равно коду ошибки; в противном случае оно будет равно нулю.
- Строка 17, которая всего лишь чуть-чуть отличается от строки 12. В ней переменной @LocalError присваивается ее текущее значение *плюс* значение функции @@ERROR; таким образом, последнее сохраняется на протяжении всей процедуры.
- Строки 18–35, в которых фактически и реализован весь механизм обработки ошибок процедуры PersonBadGuyInsert. Если значение переменной @LocalError равно нулю, значит, все в порядке и мы можем смело зафиксировать транзакцию. В противном случае следует возвратить пользователю сообщение, текст которого будет зависеть от значения переменной @LocalError (т.е. от номера ошибки). Здорово, не правда ли?

На заметку

Для того чтобы возвратить системное описание ошибки SQL Server 2000, следует воспользоваться информацией таблицы sysmessages, принадлежащей базе данных master. Описание ошибки можно извлечь с помощью обычного оператора SELECT, использовав значение функции @@ERROR в качестве критерия, например:

```
1: SELECT * FROM master.dbo.sysmessages WHERE error = @@ERROR
```

Вполне вероятно, что у вас возникли вопросы, касающиеся приведенной выше реализации механизма обработки ошибок хранимой процедуры PersonBadGuyInsert.

Во-первых, при возникновении ошибки во время выполнения первого оператора Transact-SQL значение переменной @LocalError потенциально будет равно 547 (или какому-нибудь другому отличному от нуля числу). Но если аналогичная ошибка возникнет и при выполнении второго оператора, то значение переменной @LocalError будет равно уже 1094!

Таким образом, при попытке извлечения системного описания ошибки из таблицы sysmessages, мы либо получим описание, не соответствующее нашей ошибке, либо не получим ничего вообще.

Во-вторых, при реализации механизма обработки ошибок хранимой процедуры PersonBadGuyInsert было сделано предположение о последовательном выполнении операторов Transact-SQL. Это означает, что при возникновении ошибки во время выполнения первого оператора хранимая процедура не будет тотчас же прекращена, а приступит к выполнению второго оператора. Впоследствии это можно легко изменить, особенно в том случае, если транзакция слишком долго не будет освобождать выделенные ей ресурсы.

Итак, теперь дело за вами: учитывая сказанное выше, реализуйте механизм обработки ошибок в хранимой процедуре PersonSpyInsert, изменив предварительно текст отображаемых пользователю сообщений.

Резюме

В этой главе описаны способы поддержки согласованности хранящейся в базе данных информации с использованием следующих средств:

- транзакций, которые следует применять всякий раз при необходимости изменения хранящейся в базе данных информации;
- блокировок, предотвращающих возникновение возможных внештатных ситуаций, связанных с противоречивостью данных;
- механизма обработки ошибок, который позволяет отменить изменение информации и уведомить пользователя о возникшей ошибке.

Следующие шаги

В следующей главе рассматриваются вопросы безопасности разрабатываемого приложения. В частности, вы изучите способ создания нового пользователя, познакомитесь с понятием роли в SQL Server 2000 и научитесь назначать новому пользователю необходимые привилегии.

Глава 9

Обеспечение безопасности базы данных Spy Net

В этой главе...

Наше приложение почти готово! С его помощью уже можно посылать агентов на задания. Однако необходимо учитывать, что от правильной работы приложения зависят жизни многих людей. Поэтому нужно серьезно подумать о безопасности данных приложения. В конце концов, нельзя же допустить, чтобы вражеский агент проник в нашу базу данных и вывел ее из строя или украл ценную информацию.

К счастью, SQL Server 2000 предоставляет неплохую систему безопасности, позволяющую защитить базу данных: начиная от ограничения права доступа к серверу и к базе данных и заканчивая ограничением права доступа к каждому столбцу каждой таблицы. Вот такая тщательность!

Зачем нужен такой детальный контроль над приложением? Это во многом зависит от характера доступа, который мы хотим предоставить нашим пользователям. Надежные, проверенные пользователи, которые понимают работу приложения, могут получить (и смогут использовать) больше прав, чем пользователи, слабо разбирающиеся в приложении.

Мы должны не только разработать эффективное приложение, но и тщательно изучить и учесть запросы пользователей. Только тогда наше приложение будет удовлетворять предъявляемым к нему требованиям.

Пользователи должны иметь доступ к данным в таблицах, но им незачем изменять структуру базы данных или ее объектов. Необходимо также ограничить доступ пользователей к таким базам данных на сервере, как `master`, `tempdb` и `model`, которые им не нужны.

В следующих разделах описаны типичные пользователи, обсуждаются некоторые вопросы безопасности и аудита, а также распределение прав и ролей среди наших пользователей. Итак, вперед!

Совместное использование Spy Net многими пользователями

По умолчанию после создания базы данных существует только один пользователь — `sa`. Как вы уже знаете, учетная запись `sa` не имеет ограничений.

На заметку В зависимости от установленной на компьютере операционной системы экземпляр SQL Server 2000 может иметь несколько пользователей. Например, если на компьютере установлена Windows 2000 или NT, то по умолчанию доступ к серверу разрешен пользователям встроенной группы администраторов (`Administrator`).

Учетная запись sa конфигурирована как dbo, т.е. владелец базы данных.

Термин *Владелец базы данных* (database owner) — это пользователь, который имеет право присоединять, конфигурировать, создавать и удалять указанную базу данных. Иными словами, dbo, как владелец, может делать с базой данных все, что ему захочется. Нам это подходит, однако не хотелось бы, чтобы наши пользователи, имея эти же права, могли удалить нашу базу данных. В конце концов, я не хочу после ошибки пользователя начинать все сначала.

Как защитить пользователей от них самих? Для каждого пользователя, работающего с нашей базой данных, мы создаем учетную запись и назначаем ей различные права доступа (существует или нет право выполнить некоторое действие). Эти права могут относиться к серверу, к базе данных или к объектам базы данных в текущем экземпляре SQL Server 2000. Если вам приходилось работать с Windows NT/2000, то вам хорошо знакома концепция безопасности в SQL Server 2000. Если нет — не паникуйте! Я объясню ее вам, и вы, как всегда, все отлично поймете!

Прежде чем создать первую учетную запись пользователя базы данных, рассмотрим концепцию имен пользователей. Эти имена применяются при установке соединения с экземпляром SQL Server 2000, содержащим базу данных `SQLSpyNet`. Мы использовали специальную учетную запись sa (установленную по умолчанию), поэтому до сих пор нам не понадобилось создавать новую учетную запись.

Когда пользователь пытается установить соединение с SQL Server 2000, он должен иметь учетную запись, чтобы SQL Server 2000 мог аутентифицировать его параметры. Одно из свойств учетной записи — это базы данных, к которым пользователь имеет право доступа. Сервер использует параметры пользователей для предоставления им доступа к конкретным объектам (или для отказа в доступе).

SQL Server 2000 поддерживает два режима аутентификации: режим аутентификации Windows и смешанный режим (аутентификация Windows и SQL Server). Рассматриваемый в книге экземпляр SQL Server 2000 работает на компьютере под управлением операционной системы Windows 98, который не подключен к сети, поэтому нельзя воспользоваться преимуществами аутентификации Windows. Значение этих терминов более подробно разъясняется далее в главе. Поскольку Windows 98 не поддерживает аутентификацию Windows, нам придется использовать аутентификацию SQL Server.

Создание учетных записей для SQLSpyNet

Система аутентификации SQL Server позволяет создавать собственные учетные записи пользователей, не обращаясь для этого к Windows NT/2000. Мы можем изменять пароли и управлять учетными записями независимо от операционной системы. Такой тип обеспечения безопасности предпочитают большинство разработчиков, потому что в этом случае значительно проще создавать новые учетные записи пользователей и предоставлять им соответствующие права доступа. Однако Microsoft рекомендует по возможности использовать аутентификацию Windows NT/2000.

Создадим в нашем экземпляре SQL Server 2000 новую учетную запись (в системе аутентификации SQL Server). Для этого выполните ряд действий.

1. Откройте окно Enterprise Manager, в нем найдите и откройте папку Security (Безопасность). Она содержит четыре параметра: Logins (Учетные записи), Server Roles (Роли сервера), Linked Servers (Связанные серверы) и Remote Servers (Удаленные серверы).
2. Щелкните правой кнопкой на параметре Logins и выберите команду New Login. Появится диалоговое окно, показанное на рис. 9.1.

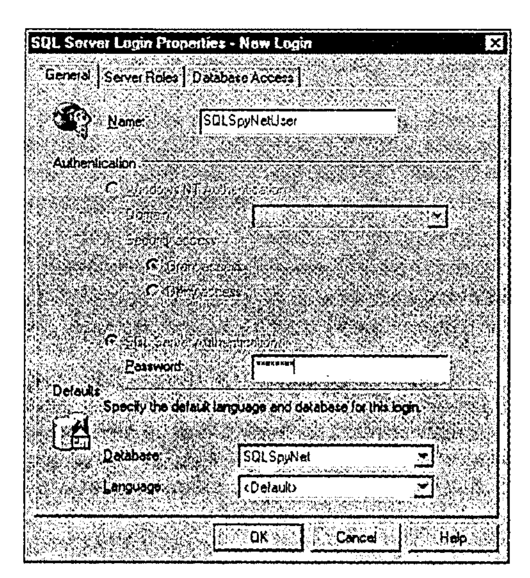


Рис. 9.1. Диалоговое окно создания новой учетной записи SQL Server 2000

Поскольку наш сервер SQL Server 2000 установлен на компьютере под управлением операционной системы Windows 98, то при выборе опции SQL Server Authentication опция Windows NT Authentication недоступна. Рассмотрим существующие способы аутентификации в SQL Server 2000.

На заметку Если в папке *Security* щелкнуть на параметре *Logins*, то можно увидеть существующие учётные записи. У нас пока создана только одна учетная запись. Но если вы работаете в Windows 2000/NT 4.0, то увидите еще одну учетную запись — `BUILTIN\Administrator`. Она позволяет администраторам локальных компьютеров устанавливать соединение в качестве владельцев баз данных SQL Server 2000 с помощью системы аутентификации Windows NT.

- *Аутентификация Windows NT.* Позволяет пользователям устанавливать соединение с SQL Server 2000, не вводя регистрационную информацию вручную. Учетная запись создана на уровне операционной системы. Она должна содержать информацию о домене и об имени пользователя, например `DISNEYLAND\Mickey`. К сожалению, этот тип аутентификации не поддерживается на компьютерах под управлением Windows 98, не подключенных к сети Windows NT/2000. Когда такой пользователь устанавливает соединение с SQL Server 2000, ему не нужно повторно вводить пароль. Режим аутентификации Windows поддерживает больше средств обеспечения безопасности, чем режим аутентификации SQL Server, например срок действия паролей или их шифрование.

- *Аутентификация SQL Server*. Сейчас мы собираемся создать учетную запись этого типа. Эта учетная запись SQL Server содержит сведения о параметрах безопасности. Учетная запись sa, которая использовалась до сих пор, основана исключительно на аутентификации SQL Server. Когда пользователь устанавливает соединение с сервером, он должен ввести имя и пароль.

На заметку Вспомните: при установке SQL Server 2000 у вас была возможность выбрать один из двух режимов аутентификации — *Windows Authentication* или *Mixed Mode*. Если вы хотите, чтобы пользователи устанавливали соединение с помощью учетных записей Windows NT/2000, то вам нужно было выбрать опцию *Windows Authentication*. Однако мы не имеем доступа к домену Windows NT, поэтому не сможем воспользоваться преимуществами аутентификации Windows.

3. Активизируйте вкладку General, в которой вводится регистрационная информация, в том числе имя учетной записи. Введите, например, **SQLSpyNetUser**. Такое имя говорит о том, что это пользователь приложения SQLSpyNet. Рекомендуется давать осмысленное имя всем создаваемым учетным записям.
4. Следующее поле в этой вкладке — пароль пользователя. Как администратор сервера (sa), вы можете присваивать и менять пароль пользователя в любое время. Пользователь тоже может изменить свой пароль. Поэтому сейчас установим в качестве пароля слово, которое пользователь легко запомнит, например **Password**.



Можно назначить для учетной записи пустой пароль (как это делается при установке по умолчанию для учетной записи sa), однако так делать не рекомендуется, потому что тогда любой пользователь сможет зарегистрироваться, зная лишь имя учетной записи. При аутентификации Windows NT можно заставить пользователя изменить свой пароль при первой регистрации.

5. Выберите для пользователя базу данных по умолчанию. Если пользователь, например, регистрируется с помощью средства Query Analyzer, то в списке баз данных для него будет выделена база данных по умолчанию. Выберите из списка в качестве базы данных по умолчанию SQLSpyNet.
6. Установите для учетной записи язык по умолчанию. Эта опция определяет характер отображения сервером некоторой информации о данной записи. Например, если выбрать British English, то дата будет выводиться в формате *dd/mm/yy*.
7. Активизируйте вкладку Database Access (рис. 9.2).

На заметку Если теперь во вкладке *General* щелкнуть на кнопке *OK* (т.е. пропустить другие вкладки), то SQL Server 2000 сначала попросит подтвердить пароль пользователя. Потом сервер выведет на экран предупреждение, сообщающее, что созданная учетная запись не имеет доступа к базе данных.

8. Выберите в списке базу данных SQLSpyNet, это даст возможность присвоить пользователю роль. Сейчас оставьте по умолчанию роль public. Далее в главе мы рассмотрим права доступа, ассоциированные с ролями.
9. Щелкните на кнопке OK. При этом SQL Server 2000 создает учетную запись, которая впоследствии отображается в окне Enterprise Manager.
10. Теперь проверьте (так нужно делать всегда), создана ли учетная запись пользователя. Для этого в базе данных SQLSpyNet откройте папку Users (рис. 9.3).

Дело сделано! Вы успешно создали свою первую учетную запись пользователя. Великолепно!

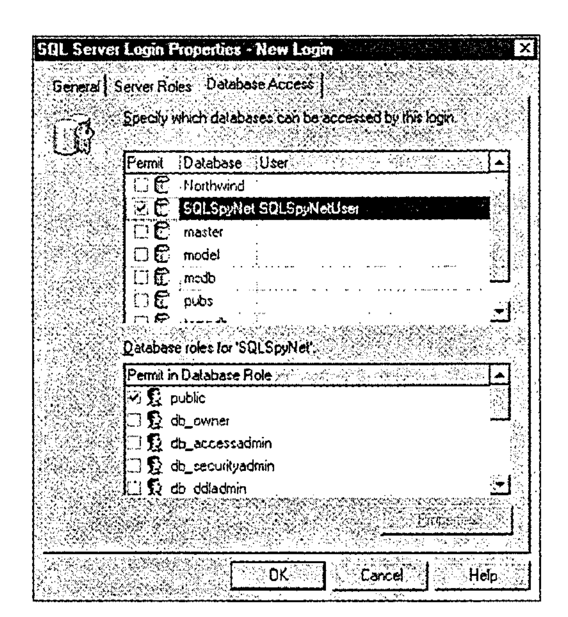


Рис. 9.2. Диалоговое окно предоставления доступа к базе данных в текущем экземпляре SQL Server 2000

Рис. 9.3. Новая учетная запись базы данных SQLSpyNet отображается в окне Enterprise Manager

Регистрация с помощью только что созданной учетной записи для проверки прав доступа

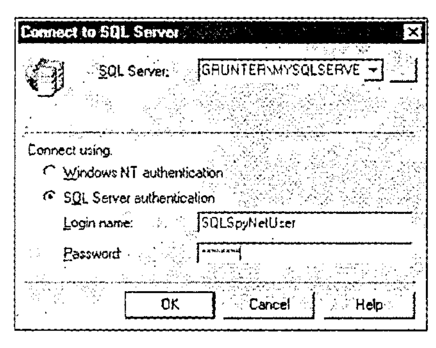


Рис. 9.4. В этом окне нужно ввести регистрационную информацию

Создавая учетную запись пользователя, мы по умолчанию присвоили ему роль `public`. Это значит, что пользователь будет иметь все права доступа, предоставленные этой роли. По умолчанию в SQL Server 2000 это довольно ограниченные права.

Сейчас наш пользователь может только устанавливать соединение с SQL Server 2000, причем только с базой данных SQLSpyNet.

Это мы сейчас и попробуем сделать! Из главного меню запустите Query Analyzer. Сервер попросит ввести регистрационную информацию (рис. 9.4). Введите **SQLSpyNetUser** и пароль. Щелкните на кнопке OK.

При этом происходит ваша регистрация в приложении под именем SQLSpyNetUser. Теперь попробуем посмотреть данные в одной из таблиц. В окне Query Analyzer выполните код листинга 9.1.

Листинг 9.1. Проверка прав доступа пользователя *SQLSpyNetUser*

Код для запуска

```
1: SELECT PersonID, Firstname, Surname FROM Person
```

Сервер возвращает сообщение об ошибке, утверждающее, что пользователь не имеет права считывать данные таблицы. Это значит также, что пользователь не может выполнять операторы `UPSATE`, `INSERT` и `DELETE`.

Теперь мы должны предоставить пользователю необходимые права доступа.

Назначение пользователям ролей

Прежде чем предоставить пользователям права доступа к приложению, рассмотрим роли, которые мы собираемся им назначить.

Экскурс

Преждевременное появление ролей

Что такое роль? Понятие ролей появилось в SQL Server 2000, когда команда разработки Windows объявила о намерении вместо термина *группы NT* применять термин *роли*. Команда SQL Server решила ввести согласующееся с операционной системой понятие роли в сервер. Однако (какой конфуз!) новая операционная система Windows 2000 вышла с группами вместо ролей. Согласование продолжается!

Понятие роли во многом аналогично понятию группы в Windows NT. Права доступа можно присвоить как роли, так и отдельному пользователю. Если права доступа назначаются роли, то каждый пользователь, включенный в эту роль, *наследует* ее права доступа.

Термин *Наследование* — это способность объекта (в данном случае пользователя) принимать все свойства другого объекта (в данном случае роли).

По умолчанию в SQL Server 2000 определены 10 базовых ролей. Они предоставляют различные уровни доступа к базе данных.

Однако существует одна специфическая роль — `public`. Каждый пользователь, определенный в базе данных, принадлежит этой роли. Она не имеет почти никаких прав доступа, кроме самых общих, например позволяющих пользователю устанавливать соединение с базой данных. Роль `public` подчиняется нескольким основным правилам.

- Она есть в каждой базе данных.
- Ее нельзя удалить.
- Роли `public` принадлежит каждый пользователь, включая `sa`.
- Поскольку каждый пользователь принадлежит этой роли по умолчанию, в нее нельзя добавить или удалить из нее пользователей.



Как отмечалось, права доступа роли `public` очень ограниченны. Их можно изменить, но будьте осторожны. Каждый пользователь принадлежит этой роли, поэтому вы можете случайно предоставить слишком много прав пользователю, который воспользуется ими неправильно.

Роли предоставляет перечисленные ниже права доступа.

- `db_owner`. С точки зрения пользователя, это божественная роль. Она предоставляет пользователю *полный* контроль над базой данных. Пользователь `sa` принадлежит роли `db_owner`, поэтому обладатель учетной записи `sa` может делать в базе данных все, что захочет.
- `db_securityadmin`. Позволяет пользователю управлять ролями и их назначением пользователям. Также позволяет назначать ролям права доступа. Если есть пользователь, которому можно доверить управление параметрами безопасности базы данных, но которому не нужен контроль над базой данных, то эту роль следует присвоить ему.
- `db_accessadmin`. Используется для предоставления пользователю права добавлять в базу данных или удалять из нее других пользователей. Как и в случае роли `db_securityadmin`, роль `db_accessadmin` обычно предоставляется пользователю, который должен управлять правами других пользователей.
- `db_ddladmin`. Позволяет пользователю манипулировать всеми объектами базы данных. Например, он может создавать, изменять и удалять объекты базы данных. Пользователи этой роли имеют право выполнять операторы языка определения данных (Data Definition Language — DDL).
- `db_backupoperator`. Предоставляет пользователю возможность выполнять резервное копирование базы данных.
- `db_datawriter`. Позволяет пользователю изменять данные во всех своих таблицах в пределах базы данных.

- db_datareader. Позволяет считывать данные во всех своих таблицах, определенных в базе данных.
- db_denydatawriter. Запрещает изменение пользователем любых данных в своих таблицах в пределах базы данных.
- db_denydatareader. Запрещает считывание пользователем данных из своих таблиц в пределах базы данных.

Пользователь может принадлежать одной роли, многим или не принадлежать ни одной. Если пользователь принадлежит более чем одной роли, его права доступа состоят из прав доступа ролей в соответствии с правилами приоритета.

Например, если пользователь SQLSpyNetUser принадлежит ролям db_datawriter и db_backupoperater, то он может не только изменять данные таблиц, но и выполнять резервное копирование базы данных.

Однако, если пользователь принадлежит ролям db_denydatawriter и db_datawriter, он не сможет изменять данные таблиц. Это объясняется тем, что отмена права доступа — более ограничительная, осторожная операция, а потому имеет более высокий приоритет, чем предоставление права доступа.

При назначении пользователям ролей нужно соблюдать определенную осторожность. Непродуманное присвоение роли может случайно отменить доступ пользователя к объекту, который ему необходим.

Присвоение ролей экземпляру SQL Server

Роли можно присваивать не только пользователям базы данных SQLSpyNet, но и пользователям всего экземпляра SQL Server 2000. Это делает процедуру установки приложений, особенно в больших узлах, более гибкой. Таким образом, можно предоставить определенным пользователям право, например, выполнять резервное копирование базы данных. Присвоение роли пользователю сервера предотвратит попытки создания резервной копии пользователями, которые недостаточно полно понимают, что они делают.

Какие роли пользователей сервера поддерживаются в SQL Server 2000?

На заметку Роли сервера вашего экземпляра SQL Server 2000 собраны в папке *Security\Server Roles*. В ней находятся все текущие роли сервера, установленные в вашем экземпляре SQL Server 2000.

- sysadmin. Пользователи этой роли имеют *полный* контроль над всем экземпляром SQL Server 2000. Этой роли принадлежит учетная запись sa.
- securityadmin. Пользователь этой роли может создавать учетные записи пользователей на сервере и управлять ими.
- serveradmin. Пользователь этой роли может конфигурировать экземпляр SQL Server 2000. Он имеет также право останавливать сервер.
- setupadmin. Пользователь этой роли имеет право управлять начальным запуском процедур и связанными серверами.
- processadmin. Пользователи этой роли имеют право управлять процессами SQL Server 2000. Это значит, что они могут выполнять команду KILL, которая прерывает сеанс пользователя. Если у пользователя есть незавершенные транзакции, то перед прерыванием сеанса они отменяются.

- diskadmin. Пользователи этой роли имеют право управлять файлами на диске, в том числе группами файлов (см. главу 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet").
- dbcreator. Пользователи этой роли могут создавать, изменять и удалять базы данных.

На заметку Большинство этих ролей позволяют добавлять другие учетные записи пользователей. Например, если запись SQLSpyNetUser принадлежит роли dbcreator, то данный пользователь может добавлять в эту роль любою другою учетную запись.

Однако добавлять новые учетные записи позволяют не все роли, например db_datawriter.

Итак, роли позволяют выполнять некоторые действия; их мы сейчас и обсудим.

Что, если эти роли не подходят для решения стоящих перед пользователями задач? Рассмотрим, как создавать и конфигурировать собственные роли.

Модель роли

Допустим, есть 10 новых пользователей, которым для выполнения их задач встроенные роли сервера или базы данных не совсем подходят. Что можно сделать в этом случае?

SQL Server 2000 позволяет создавать собственные роли. Если все пользователи должны иметь одинаковые права доступа, то оптимальным решением будет следующее: создать новую роль, добавить к ней каждую учетную запись пользователя и предоставить или отменить права доступа к каждому объекту, с которым они будут работать.

Звучит великолепно! Идеальное решение проблемы!

Однако в данный момент у нас есть только один пользователь. Стоит ли создавать целую роль всего лишь для одного пользователя?

К счастью, SQL Server 2000 позволяет предоставлять или отменять право доступа каждого отдельного пользователя к каждому отдельному объекту базы данных. Например, мы можем предоставить пользователю SQLSpyNetUser право изменять адреса в таблице Address (права SELECT, INSERT, UPDATE и DELETE), но запретить ему изменять какие-либо данные любой другой таблицы.

На заметку Такая гибкость доступна в SQL Server 2000 (и фактически во всех версиях SQL Server), однако SQL Server 2000 позволяет даже отменить право SQLSpyNetUser на просмотр отдельного столбца таблицы Spy, например столбца AnnualSalary. Мы можем очень детально контролировать права доступа пользователей.

В следующих главах будет добавлена еще одна учетная запись пользователя базы данных SQLSpyNet. Он будет предназначен для Web-интерфейса пользователя, который мы вскоре создадим.

Сейчас создадим для базы данных новую роль и присвоим ее учетной записи нашего нового пользователя. Этой роли мы присвоим базовые права доступа, которые позволят ее пользователю выполнять все необходимые операции над базой данных.

Сначала создадим роль базы данных.

1. Откройте окно Enterprise Manager и в базе данных SQLSpyNet выделите объект Roles.
2. Щелкните правой кнопкой мыши на этом выделенном объекте и выберите команду New Database Role (Новая роль базы данных). Появится диалоговое окно, показанное на рис. 9.5.

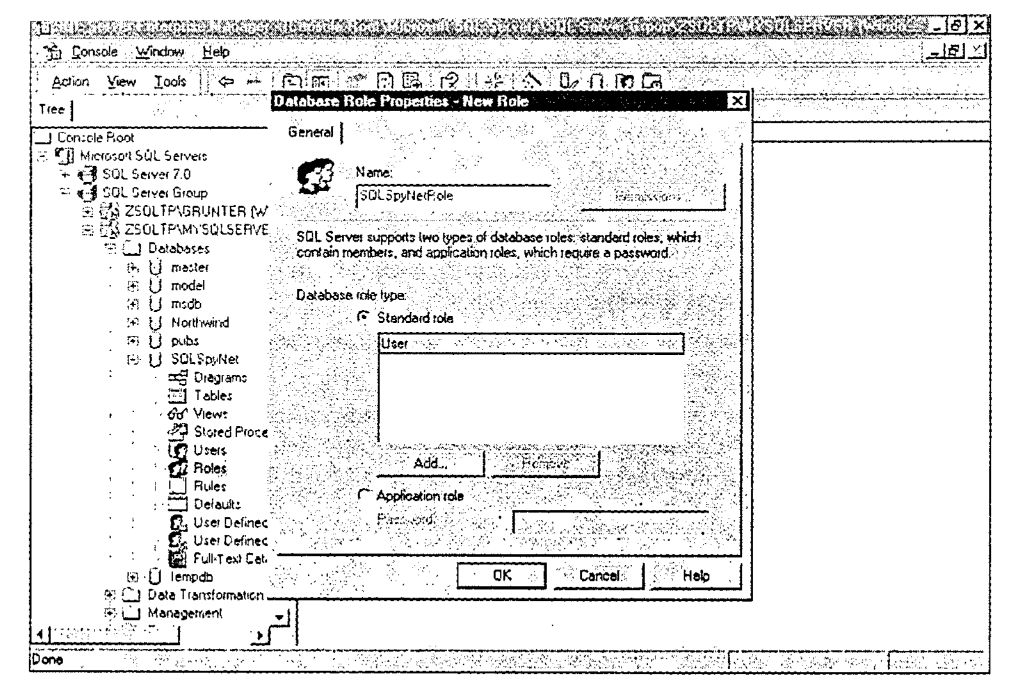


Рис. 9.5. Создание новой роли базы данных SQLSpyNet

3. Как и для большинства объектов базы данных, роли нужно дать имя, которое SQL Server 2000 будет использовать для идентификации этого объекта. Назовите ее SQLSpyNetRole. Роли можно присвоить любое имя, однако гораздо лучше, если оно будет описательным.
4. Установите переключатель Standard role (Стандартная роль). Обратите внимание, что SQL Server предлагает два типа ролей.

- Standard role. Роли этого типа похожи на группы Windows NT. Они могут содержать учетные записи пользователей, которым могут быть назначены права доступа. Когда пользователь такой роли устанавливает соединения, они наследуют права доступа роли (а также все другие права доступа пользователя).
- Application role (Роль приложения). Роли этого типа принадлежат исключительно SQL Server 2000, они не содержат учетных записей пользователей. Их можно представить как учетные записи пользователей. Роли приложения выступают в качестве пользователей, потому что для их активизации необходим пароль. Однако, когда роль активизирована (с помощью пароля и хранимой процедуры `sp_setapprole`), соединение теряет все права доступа. Теперь аутентификацию пользователей должно выполнять приложение. Однако, поскольку при установке соединения SQL Server 2000 должен аутентифицировать приложение, оно должно предоставить пароль.

5. Добавим в эту роль пользователя SQLSpyNetUser. Для этого щелкните на кнопке Add. Появится окно, показанное на рис. 9.6.
6. Это окно содержит список текущих пользователей базы данных. Поскольку у нас только один пользователь, мы видим только его. В этом окне нужно выделить пользователей, которых необходимо добавить в роль, и щелкнуть на кнопке OK. Так и сделайте.

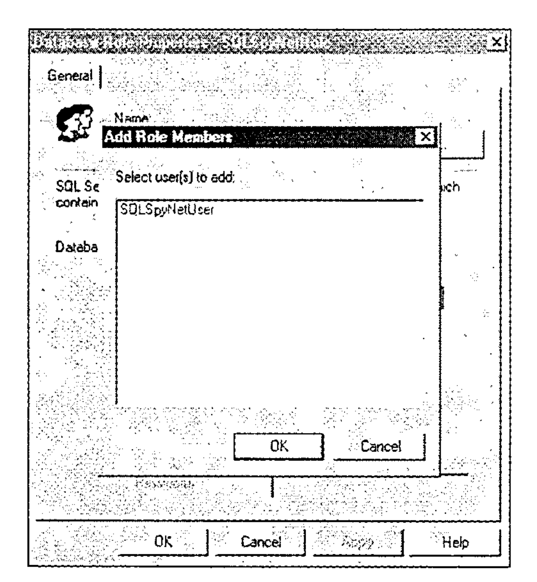


Рис. 9.6. Присвоение пользователю SQLSpyNetUser роли SQLSpyNetRole

Итак, наш пользователь теперь принадлежит роли SQLSpyNetRole и наследует все ее права доступа (правда, у нее пока никаких прав нет).



Пока не закрывайте окно *Database Roles Properties*. Из него можно открыть окно *Permissions* (Права доступа), которыми мы займемся в ближайшее время.

Теперь предоставим роли SQLSpyNetRole некоторые основные права доступа к базе данных. Это можно сделать двумя способами: выполнив соответствующий сценарий SQL или воспользовавшись инструментами Enterprise Manager. Сейчас воспользуемся графическими инструментами Enterprise Manager.

Выбор прав доступа для конкретной задачи

Прежде чем приступить к назначению нашей роли прав доступа, необходимо четко представить, какие операции должны выполнять пользователи этой роли. Да, вы правы! Наши пользователи — это пользователи всей системы. Они должны иметь возможность просматривать и редактировать некоторые (однако не все) данные.

Что они могут делать?

- Вставлять, обновлять и удалять адреса.
- Вставлять, обновлять и удалять страны.
- Вставлять, обновлять и удалять типы адресов.
- Вставлять и обновлять данные таблицы `Person`.
- Вставлять и обновлять данные представления `PersonAddress`.
- Выполнять функции форматирования даты.

Что они не должны делать?

- Просматривать или изменять данные таблиц `Spy` и `BadGuy`.

- Просматривать или изменять данные таблицы ActivityType.
- Просматривать или изменять данные таблицы Activity.
- Выполнять хранимые процедуры PersonBadGuyInsert и PersonSpyInsert.
- Изменять структуру данных.
- Создавать новые объекты базы данных.
- Выполнять задачи администрирования.

Есть что-нибудь еще? Вполне возможно. В процессе расширения нашего приложения можно пересмотреть ограничения, накладываемые ролями. В этом случае можно легко скорректировать права доступа ролей. Однако на данный момент, кажется, учтено все.

Как предоставить роли (и пользователю) эти права доступа?

Предоставление прав доступа

Теперь предоставим роли определенные ранее права доступа.

На заметку Если не задать права доступа роли к конкретным объектам базы данных, то пользователи роли не получат доступа к этим объектам.

1. Если вы не закрыли окно Database Role Properties, щелкните на кнопке Permissions. (Если все же закрыли, то сначала откройте его, дважды щелкнув на объекте SQLSpyNetRole в папке Roles). Появится окно, показанное на рис. 9.7.

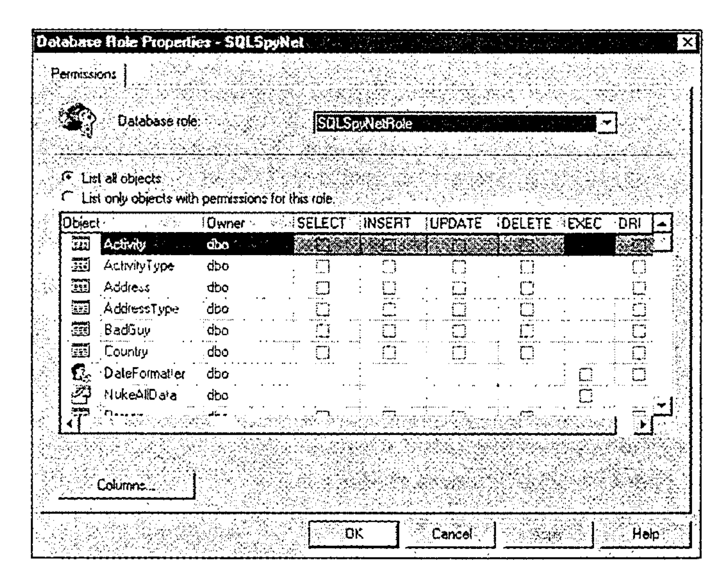


Рис. 9.7. Окно предоставления прав доступа к объектам базы данных для роли SQLSpyNetRole

В этом окне представлен список объектов базы данных и все права, которые можно присвоить роли SQLSpyNetRole. Как видите, пока что наши пользователи не имеют никаких прав, кроме права установить соединение с базой данных.



Чтобы увидеть, какие права имеет роль (или пользователь) в этой базе данных, установите переключатель *List only objects with permissions for this role* (Список объектов, на которые эта роль имеет права).

2. Чтобы предоставить роли права, которые мы считаем для нее необходимыми, щелкните в каждом нужном квадратике, соответствующем праву доступа SELECT, INSERT, UPDATE или DELETE (рис. 9.8).

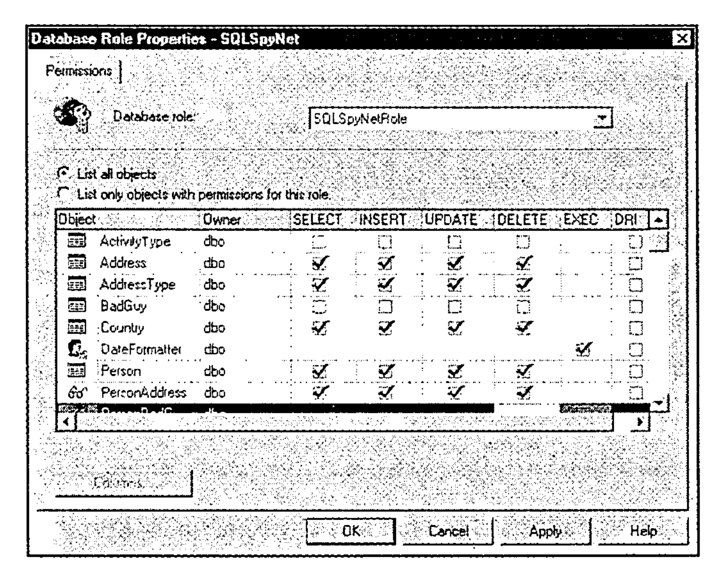


Рис. 9.8. Присвоение прав роли SQLSpyNetRole для каждого объекта базы данных

На заметку

Сейчас вы уже, конечно, понимаете, что *SELECT*, *INSERT*, *UPDATE* и *DELETE* — это права доступа. Однако с правами *EXEC* и *DRI* вы пока не знакомы. Право *EXEC* позволяет выполнять хранимые процедуры, а *DRI* означает декларативную ссылочную целостность, оно позволяет изменять структуру отношений между объектами базы данных.

3. Назначьте для нашей роли права доступа, как показано в табл. 9.1.

Таблица 9.1. Права доступа роли *SQLSpyNetRole*

Объект	*SELECT*	*INSERT*	*UPDATE*	*DELETE*	*EXEC*	*DRI*
`Address`	X	X	X	X		
`AddressType`	X	X	X	X		
`Country`	X	X	X	X		
`DateFormatter`					X	
`Person`	X	X	X	X		
`PersonAddress`	X	X	X	X		

4. Щелкните на кнопке Apply (Применить). Теперь роль имеет все необходимые права.

Обратите внимание на кнопку Columns в нижней левой части окна. С ее помощью можно предоставить права SELECT и UPDATE для отдельных столбцов базы данных. Таким образом, мы можем детально определить, что могут делать пользователи.



Если кнопка *Columns* недоступна, щелкните на одной из таблиц в списке, например на таблице *Address*. Кнопка станет активной. Щелкните не ней, и вы увидите список столбцов таблицы.

Превосходно, не так ли? Но какие еще права мы можем предоставить (или отменить) нашим пользователям и ролям? Об этом — во второй половине следующего раздела.

Проверка прав доступа к базе данных

Мысленно вернитесь к главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet", в ходе которой была создана база данных SQLSpyNet. В этой главе упоминалась вкладка Permissions, однако никаких прав доступа для пользователей не устанавливалось, потому что тогда еще не было пользователей.

Теперь, когда у базы данных есть пользователи, мы можем просмотреть различные права доступа к ней. Например, можно посмотреть, какие пользователи имеют право изменять данные или выполнять резервное копирование базы данных.

Чтобы увидеть права доступа к *базе данных*, выполните следующее.

1. Откройте окно Enterprise Manager и щелкните на базе данных SQLSpyNet.
2. Щелкните правой кнопкой мыши и из контекстного меню выберите команду Properties.
3. В появившемся окне активизируйте вкладку Permissions (рис. 9.9).

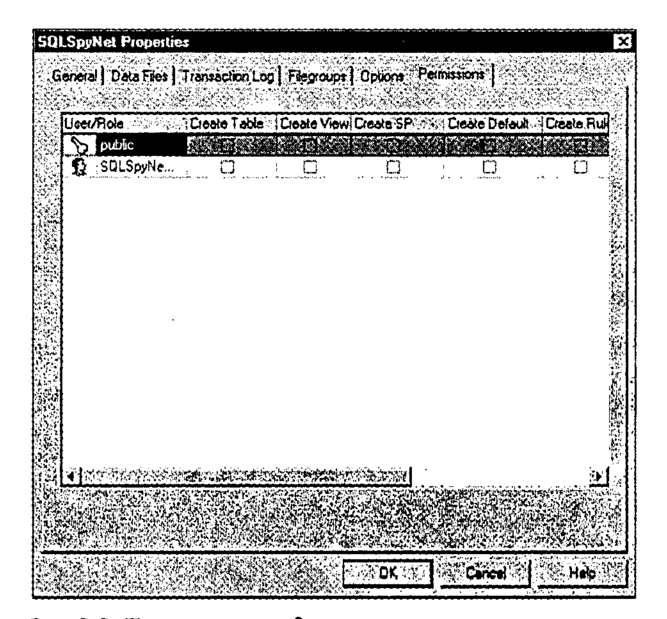


Рис. 9.9. Просмотр прав доступа различных пользователей к базе данных SQLSpyNet

В списке вы видите учетную запись пользователя SQLSpyNetUser. Ей можно предоставить право создавать объекты базы данных, например представления, таблицы и т.д.

Можно также предоставить пользователям, принадлежащим некоторым ролям, право выполнять административные задачи. Однако пока не предоставляйте нашему пользователю административных прав. В будущем, когда мы расширим состав пользователей, более опытным можно предоставить права администрирования, например право создавать и восстанавливать резервную копию базы данных. В конце концов, вы ведь не хотите делать всю административную работу сами!

Использование предоставленных прав

Теперь наш пользователь имеет некоторые права. Как это проверить? Проще всего это сделать, зарегистрировавшись в Query Analyzer с этой учетной записью. Таким образом, можно стать на место пользователя и проверить, какие права мы имеем.

Откройте окно Query Analyzer. Начинаем тестирование! Зарегистрировавшись в Query Analyzer как пользователь SQLSpyNetUser, выполним некоторые тестовые программы, чтобы удостовериться, что мы не сможем делать того, что нельзя, но сможем делать все, что необходимо для нашей работы (процедура регистрации описана выше в главе).

В окне Query Analyzer выполните код листинга 9.2.

Листинг 9.2. Проверка прав доступа учетной записи SQLSpyNetUser роли SQLSpyNetRole к таблице Person

Код для запуска

```
1: SELECT PersonID, Firstname, Surname FROM Person
```

Сервер должен вернуть все записи таблицы Person (рис. 9.10).

Теперь попробуем сделать то же для таблицы Spy. К этой таблице наша роль не имеет права доступа. В окне Query Analyzer выполните код листинга 9.3.

Листинг 9.3. Проверка доступа учетной записи SQLSpyNetUser к таблице Spy

Код для запуска

```
1: SELECT SpyID, PersonID, SpyNumber, Alias FROM Spy
```

Этот запрос должен возвратить ошибку, сообщающую, что пользователь не имеет права выполнять для таблицы оператор SELECT (рис. 9.11).

На заметку Если вы увидели возвращенными все записи таблицы Spy, то сначала проверьте, с какой учетной записью вы зарегистрировались в *Query Analyzer*. Чтобы увидеть свое текущее регистрационное имя, выполните оператор SELECT SYSTEM_USER. Если имя верное, проверьте права доступа, предоставленные на предыдущих шагах.

Итак, права доступа предоставлены. Мы проверили их и убедились, что они назначены правильно. Однако следует пойти дальше и проверить права доступа для остальных объектов базы данных.

Мы можем предоставлять и отменять права доступа ролей и пользователей, однако можем ли мы увидеть, что они делают в нашей базе данных? Да! Это можно сделать с помощью аудита.

```
SELECT PersonID, Firstname, Surname FROM Person
```

	PersonID	Firstname	Surname
1	1	Richard	Darstidly
2	2	Druisilla	Sharp
3	3	Gertrude	Heiglemeitster
4	4	Boris	Kremlin
5	5	Wingtong	Tiddle-I-Po
6	6	Phillip	McKracken
7	7	Josh	Anderson
8	8	Zsolt	Bolokov

Query batch comp ZSOLTP\MYSQLSERVER (8.0) SQLSpyNetUser (52) SQLSpyNet 0:00:00 21 rows Ln 1, Col 48

Рис. 9.10. Демонстрация правильной установки прав доступа `SQLSpyNetUser` *к таблице*

```
SELECT SpyID, PersonID, SpyNumber, Alias FROM Spy
```

```
Server: Msg 229, Level 14, State 5, Line 1
SELECT permission denied on object 'Spy', database 'SQLSpyNet', owner 'dbo'
```

Query batch compl ZSOLTP\MYSQLSERVER (8.0) SQLSpyNetUser (52) SQLSpyNet 0:00:00 0 rows Ln 1, Col 50

Рис. 9.11. Сервер отказал пользователю `SQLSpyNetUser` *в праве просмотра записей таблицы* `Spy`

Аудит: "Большой брат" на чеку!

Сервер SQL Server 2000 представляет собой инструмент всеобъемлющего контроля базы данных. С его помощью можно осуществлять аудит текущего экземпляра SQL Server 2000. Это значит, что мы можем отслеживать все действия на сервере: неудачные попытки регистрации, предоставление и отмену прав доступа, резервное копирование и восстановление и т.д. В журнал аудита можно даже записать события запуска и остановки самого процесса аудита.

Зачем нужен аудит экземпляра SQL Server 2000? Чтобы отследить потенциальные проблемы, которые могут появиться из-за неверных прав доступа или ошибок пользователей и пользовательских программ.

Объединив средства аудита SQL Server 2000 и SQL Profiler, можно проследить действия пользователей с момента их регистрации вплоть до разрыва соединения.

В SQL Server 7.0 тоже были средства аудита, однако в SQL Server 2000 они значительно усовершенствованы. Когда аудит включен, сервер ведет себя несколько иначе, чем в том случае, когда он выключен. При включении аудита SQL Server 2000 создает журнал аудита, размер которого по умолчанию равен 200 Мбайт. Файл этого журнала создается в момент запуска сервера, поэтому сервер запускается немного дольше, чем обычно.

Если файл не может быть создан из-за нехватки дискового пространства, SQL Server 2000 останавливает свою службу (MSSQLSERVER), т.е. пользователи не смогут установить соединение с сервером.

Зачем нужна такая жесткая система безопасности? SQL Server 2000 поддерживает две модели аудита: базовый и аудит C2. Первая модель позволяет перехватывать основные события пользователей. Вторая несколько жестче первой. Сначала рассмотрим базовый аудит.

Базовый аудит в SQL Server 2000

Сервер SQL Server 2000 позволяет перехватывать многие события, которые пользователи выполняют в базе данных и в экземпляре SQL Server 2000. Как это достигается? Аудит можно проводить с помощью двух средств SQL Server 2000: SQL Profiler (описан в главе 2, "Компоненты SQL Server 2000") и встроенных инструментов аудита в Enterprise Manager.

SQL Profiler — инструмент весьма разносторонний. С его помощью можно перехватывать многие события, выполняемые пользователями в базе данных. Он позволяет отследить фактически все, что в ней происходит! Можно также определить параметры конфигурации экземпляра SQL Server 2000, в которых необходимо перехватить информацию аудита. Можно перехватывать успешные регистрации пользователей на сервере, неудачные попытки регистрации и т.д.

Зачем же может понадобиться перехват событий, выполняемых пользователями в базе данных?

Данные очень чувствительны. Особенно в нашем примере. С помощью ODBC установить соединение с базой данных сравнительно несложно. Установив соединение, пользователь-хакер может попытаться получить доступ к секретной информации, содержащейся в наших таблицах.

Что будет, если злоумышленнику удастся получить доступ к нашей информации? Он может сорвать выполнение всех наших операций, и мир останется не защищенным от террористов!

С помощью эффективной стратегии безопасности мы можем ограничить круг объектов, доступных для пользователя, как сделали это для учетной записи

`SQLSpyNetUser`, в то время как с помощью аудита можно контролировать все действия пользователей!

Однако что делать, если базового аудита недостаточно? К счастью, SQL Server 2000 предоставляет тип аудита, пригодный даже для такой чувствительной организации, как наша. Рассмотрим следующий шаг в наращивании возможностей аудита.

Аудит C2

Аудит C2 основан на одноименном стандарте, утвержденном Министерством обороны США. Представьте себе — это весьма серьезно! Перехватываются не только соединения пользователя, но и *любые* действия, затрагивающие безопасность, например создание и удаление учетных записей, ролей и т.д.

Операционная система Windows 2000 соответствует стандарту C2. Более подробную информацию о нем можно найти на Web-узле Radium по адресу: `http://www.radium.ncsc.mil/tpep/epl/entries/TTAP-CSC-EPL-00-001.html`.

Аудит C2 способен перехватывать все события, которые могут нарушить безопасность компьютерной системы. Государственный стандарт C2, определяющий уровень безопасности системы, регламентирует следующее.

- Ограничение или предоставление прав доступа к базе данных или серверу как для отдельных пользователей, так и для групп. SQL Server 2000 обеспечивает это с помощью учетных записей и ролей.
- Наличие возможности уникальной идентификации пользователей системы. SQL Server 2000 не позволит создать две одинаковые учетные записи, поэтому пользователь системы может быть однозначно идентифицирован в любой точке.
- Предоставление и отмену прав доступа к ресурсам экземпляра SQL Server 2000 должен осуществлять только администратор системы.

Это, однако, далеко не полный список. Как вы понимаете, Министерство обороны США не отнеслось к этому легкомысленно.

Мы пока еще не готовы выполнить аудит, потому что разработка нашего приложения не завершена, однако это время не за горами. В главе 14, "Отладка и устранение ошибок в SQL Server 2000", будут рассмотрены все необходимые для этого средства. Вы получите базовые знания по вопросам установки и проведения аудита для обеспечения безопасности ваших приложений.

Разработка стратегии безопасности

Можно ли обеспечить высокий уровень безопасности, существенно не усложняя задачи администрирования? Чтобы достичь этого, необходимо учесть ряд соображений.

- Чтобы уменьшить трудоемкость выполнения задач администрирования, предстоящих вам в будущем, используйте роли. Когда вы определите роль (а вы можете определить их хоть сотни), вам останется только добавлять и удалять учетные записи этой роли. В этом случае вам не придется заниматься правами доступа каждого пользователя персонально.
- Безопасность данных в SQL Server 2000 чрезвычайно важна. Поэтому перед выполнением *каждой* операции над *каждым* объектом базы данных проверяются права доступа пользователя к этому объекту. Это же относится и к учетной записи `sa`! Такой подход обеспечивает полное соблюдение требований принятой модели безопасности.

- Мы можем предоставлять или отменять права доступа пользователей к объектам базы данных. Если же пользователь имеет право создать представление или хранимую процедуру, может ли он включить в них таблицу, к которой у него нет права доступа? Да. Пользователь может *создать* объект, если он имеет право создать его. Но, попытавшись *выполнить* эту хранимую процедуру или представление, он получит сообщение об ошибке, утверждающее, что у него нет права доступа к таблице или другому объекту, используемому этой хранимой процедурой.

  Следовательно, требования модели безопасности будут соблюдены, даже если пользователь окажется настолько хитрым. Как это происходит? База данных проверяет права доступа на каждый объект, к которому обращается пользователь, поэтому она находит в хранимой процедуре таблицы, к которым пользователь не имеет права обращаться. Весьма предусмотрительно, не так ли?

Что еще нужно знать об аудите? Массу вещей! Концепции аудита довольно просты, однако проверка действий пользователей содержит много тонкостей. Поэтому необходимо учитывать следующее.

- Когда запускается SQL Server 2000, он пытается создать журнал аудита. Его файл имеет размер 200 Мбайт — ни больше, ни меньше! Таким образом, несколько запусков сервера могут легко переполнить диски. Если свободного дискового пространства окажется недостаточно, то сервер не запустится, потому что не сможет выполнять аудит. Поэтому у вас должен быть хороший план архивирования журналов аудита. Нет смысла выполнять их лишь для того, чтобы заполнить диск.
- Аудит уменьшает эффективность работы базы данных. Каждое событие, порождаемое пользователями, перехватывается и записывается на диск, что, естественно, уменьшает скорость выполнения запросов. В системах, поддерживающих сотни одновременно работающих пользователей, это замедление весьма существенно. Причем аудит C2 снижает производительность сервера гораздо больше, чем базовый, поскольку должны протоколироваться еще и действия, связанные с обеспечением безопасности.
- Зачем нужно протоколировать события? Было бы смешно перехватывать все события и записывать их в журнал, никак не используя эту информацию. Если аудит проводить часто и количество пользователей велико, то журналы аудита будут содержать огромное число записей. Придется дополнительно нанять работника специально для того, чтобы он просматривал эти журналы!

Резюме

Итак, дамы и господа, закончилась очередная глава. Сделан еще один шаг, приблизивший нас к заветной цели — завершению проекта!

Были рассмотрены типы аутентификации, предоставляемые SQL Server 2000, а также обеспечение различных уровней безопасности путем установки соответствующих прав доступа. Описаны типы аудита и возможности SQL Server 2000 в этом отношении.

В конце главы кратко перечислены некоторые факторы выбора стратегии безопасности при разработке приложения. Средства обеспечения безопасности — это последнее, что вам придется реализовать при разработке базы данных, но далеко не

последнее по степени значимости для успешной работы всего приложения. Очень важно правильно определить модель обеспечения безопасности. На это придется потратить некоторое время, однако в будущем вы не пожалеете об этом!

Следующие шаги

В следующей главе рассматривается обеспечение доступности данных для пользователей в любой момент, когда они им понадобятся. Описываются некоторые наиболее общие задачи администрирования баз данных, такие как резервное копирование данных и их восстановление в случае ошибок. Итак, вперед, господа!

Глава 10

Обеспечение доступности данных

В этой главе...

Опять пришла пора сменить роль! Мы уже не раз это делали. Начав как аналитики, мы стали разработчиками приложений баз данных, а затем — экспертами в вопросах безопасности. Теперь самое время выступить в роли администратора баз данных.

Ручаюсь, вы и не думали, что придется освоить так много специальностей. Но, возможно, рано или поздно вы придете наниматься на должность разработчика. И тогда вам предстоит узнать еще немало нового.

Вернемся к администрированию баз данных. В обязанности администратора входит выполнение многих задач, обеспечивающих правильное и эффективное функционирование баз данных. До сих пор изложенный материал был ориентирован главным образом на разработку проектов, однако в этой главе мы будем иметь дело с вопросами администрирования. Прежде чем разрабатывать базы данных, нужно хорошо изучить стратегию их функционирования.

Как обеспечивается доступность базы данных? Невозможно гарантировать, что наша база данных всегда будет полностью работоспособной, однако при возникновении ошибок или выходе какого-либо узла из строя мы должны уметь быстро восстановить ее работоспособность.

Как администраторы баз данных мы отвечаем перед разработчиками, руководством фирмы и клиентами за сохранность данных и работоспособность приложения при любых неблагоприятных событиях.

Каким образом это достигается? С помощью тщательно продуманного плана восстановления базы данных в аварийных ситуациях. План основывается на операциях резервного копирования и восстановления, предоставляемых сервером SQL Server 2000. С их помощью можно скопировать базу данных в целостном состоянии и, если произойдет авария оборудования или катастрофические программные ошибки, после восстановления сервера быстро восстановить ее доаварийное состояние. Базовые элементы эффективной стратегии доступности данных включают следующее:

- заблаговременное планирование;
- понимание принципов работы журнала транзакций и умение использовать его для восстановления данных;
- резервные копирования базы данных;
- восстановление данных.

В этой главе рассматриваются базовые концепции, лежащие в основе создания плана, а также способы создания резервных копий и восстановления базы данных.

В конце главы я предложу вам небольшой список, на основе которого вы сможете построить собственный план резервного копирования и восстановления.

Стратегия резервного копирования базы данных

Резервное копирование входит в обязанность администратора базы данных. Без эффективной стратегии резервного копирования можно попасть в ситуацию, когда база данных выходит из строя, а достаточно новой резервной копии для ее восстановления не существует. Для выработки эффективной стратегии нужно понимать имеющиеся способы резервного копирования, уметь правильно выбрать периодичность копирования и правильно определить место хранения драгоценных файлов.

Эффективная стратегия резервного копирования позволяет восстановить базу данных при многих видах аварий, включая следующие:

- неправильный ввод данных пользователем, например ввод команды завершения месяца в его начале;
- крах жесткого диска;
- крах сервера.

Подобные ситуации возникают не каждый день, однако иногда они все же происходят, поэтому нужно быть готовым к худшему.

Резервное копирование предоставляет еще одну интересную возможность (причем бесплатно!). Мы можем использовать резервную копию базы данных SQL Server 2000 (или какой-либо иной) для быстрого переноса базы данных на совершенно другой сервер. При этом нет необходимости заново создавать все объекты базы данных, потому что они, как и данные, содержатся в резервной копии.



Если не обеспечить безопасность данных, то с помощью этого средства SQL Server 2000 кто угодно сможет положить полную копию драгоценных данных в карман и унести с собой. Вот почему в главе 9, "Обеспечение безопасности базы данных Spy Net", мы предложили ограничить для ролей права копирования.

Как предотвратить потерю всего

Эффективная стратегия резервного копирования позволяет восстановить базу данных после аварии. Но какую стратегию следует считать эффективной? Ведь эффективность стратегии в полной мере проявится только после аварии. Значит, для определения ее эффективности нужно ждать аварии? Конечно, нет! Вы должны предвидеть возможный характер различных типов аварий и обеспечить восстановление при любой из них. При выработке эффективной стратегии придерживайтесь следующих рекомендаций:

- выполняйте резервное копирование регулярно (конкретная периодичность зависит от условий работы базы данных);

- защищайте вашу резервную копию с помощью паролей; держите пароли под замком в прочном сейфе;
- храните *полные* копии базы данных в другом месте;
- регулярно выполняйте проверки целостности базы данных;
- тщательно планируйте резервное копирование;
- рассчитывайте на худшее!

Приведенный список далеко не исчерпывающий. Создание эффективной стратегии резервного копирования предполагает многочисленные консультации с другими группами. Фактически каждый, кто каким-либо образом зависит от данных (как, например, держатели акций), нуждается в информации об этих данных. Только при наличии достаточной информации можно оценивать запросы пользователей и строить на этой основе правильную стратегию. К счастью, пока что мы можем заботиться только о нас самих!

Использование журналов транзакций при резервном копировании и восстановлении

Сервер использует журнал транзакций для отслеживания всех обновлений, вставок и удалений, выполняемых в базах данных. Для каждой базы данных SQL Server по умолчанию создает и поддерживает один журнал транзакций (однако при необходимости их может быть больше).

Концепция журнала транзакций чрезвычайно важна в SQL Server 2000. В нем хранятся записи *изменений* в базе данных, поэтому с его помощью можно (наряду с резервными копиями) восстановить целостное состояние базы данных.



Объем журналов транзакций со временем увеличивается, поэтому они нуждаются в поддержке. В главе 14, "Отладка и устранение ошибок в SQL Server 2000", приведены некоторые советы относительно того, как предотвратить заполнение журналами транзакций всего доступного дискового пространства.

Экскурс

Журналы, журналы, кругом одни журналы

Сейчас вы, наверное, удивляетесь разнообразию типов журналов в SQL Server 2000.

В главе 9, "Обеспечение безопасности базы данных Spy Net", рассматривались журналы аудита. Этот тип журналов используется для перехвата действий пользователей в экземпляре SQL Server 2000. Обычно их используют для управления безопасностью данных. Например, если пользователь (администратор) добавляет в роль базы данных новую учетную запись, конфигурирует сервер или запускает службу MSSQLSERVER, то эти действия записываются в журнал аудита.

В главе 11, "Администрирование разведывательной сети", рассматривается журнал активности. Он похож на журналы событий Windows NT/2000 и отслеживает действия, выполняемые в экземпляре SQL Server 2000. Например, если запускается служба MSSQLSERVER, то это событие записывается в журнал. Если возникает ошибка, она также записывается в журнал активности.

В этой главе рассматриваются журналы транзакций. Они отслеживают все изменения данных, выполненные в базе данных. Например, когда Салли обновляет адрес Ричарда Дастидли, в журнал транзакций записывается оператор `UPDATE`.

Экскурс

Выгода использования журналов транзакций заключается в том, что когда база данных потерпит крах (рано или поздно это все же случится), с помощью резервной копии можно восстановить состояние базы данных, в котором она пребывала на момент копирования (скорее всего, последнее копирование выполнялось прошлой ночью), а затем, выполняя транзакции, сохраненные в журнале транзакций, восстановить как можно более новое состояние базы данных.

Выбор модели восстановления

Теперь для вас очевидно, зачем нужно выполнять резервное копирование. Но какие типы восстановления можно использовать? SQL Server 2000 поддерживает три модели восстановления.

На заметку

Когда я использую термин *журнал*, я подразумеваю журнал транзакций, который содержит все транзакции и все изменения базы данных, осуществленные этими транзакциями.

- *Полная.* Из трех эта (как следует из названия) наиболее полная. Если терпит крах жесткий диск, она позволяет восстановить состояние базы данных не только на момент краха, но и на любой заданный момент времени. Чтобы это было возможным, в журнал протоколируются все операции. Но, поскольку протоколируются все действия, журнал растет очень быстро.

Термин

Восстановление *на любой момент времени* означает возможность восстановления базы данных в том состоянии, в котором она находилась в любой заданный момент времени (не слишком давно, конечно). Это чрезвычайно мощное средство, оно позволяет поддерживать для коммерческой базы данных стандарт доступности данных 24 часа в сутки в течение 7 дней в неделю. При этом потенциальная потеря изменений в базе данных настолько мала, насколько это вообще возможно физически.

Термин

Восстановление *на момент краха* означает восстановление состояния базы данных, которое она имела в момент краха системы (при условии, что журнал транзакций существует и не поврежден).

- *Массовая.* Эта модель предполагает полное резервное копирование базы данных, однако записи транзакций массовых операций в журнале минимальны, поэтому журнал не заполняется так быстро, как при полной модели. Если терпит крах жесткий диск, то массовая модель позволяет восстановить данные на момент краха, но не на любой момент времени.
- *Простая.* Это наиболее "легковесная" модель. Она требует наименьшего дискового пространства (размер журналов минимален) и наименее интенсивно использует ресурсы. Однако возможности потери данных повышаются. Эта модель не позволяет восстановить базу данных как на любой момент, так и на момент краха. Сейчас эта модель используется для нашей базы данных SQLSpyNet.

Каждая из этих моделей имеет достоинства и недостатки. Какую из них выбрать? Это зависит от характера решаемых задач. Если в базе данных интенсивно выполняются транзакции, а восстановление должно быть полным и на любой момент, то

наиболее подходящей будет полная модель. Если в базе данных выполняется много массовых вставок (таких, как SELECT INTO), а восстановление индивидуальных транзакций пользователей не актуально, то следует выбрать массовую модель.

Простая модель предназначена для некритичных систем, например для приложения в процессе разработки. В этой модели восстановление на момент краха не поддерживается, поэтому при ее использовании приходится восстанавливать состояние базы данных на момент последнего копирования и просить пользователей ввести данные повторно. Сейчас, разрабатывая базу данных SQLSpyNet, мы используем простую модель. Когда модель восстановления потребуется нам для реального дела, нужно будет перейти к полной или, как минимум, массовой модели.

Не беспокойтесь, вскоре вы узнаете, как перейти к другой модели.

Проверка текущего уровня восстановления базы данных SQLSpyNet

Уровень (модель) восстановления базы данных задается при ее создании. Эта опция, как и многие другие опции конфигурирования, устанавливается в базе данных model, которая рассматривалась при создании базы данных Spy Net (глава 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet").

Чтобы узнать модель восстановления базы данных, нужно выполнить функцию databasepropertyex (листинг 10.1). Это новая встроенная функция SQL Server 2000, она возвращает довольно много полезной информации о базе данных. Более подробно эта функция рассматривается в справочной системе Books Online.

Листинг 10.1. Получение информации о модели восстановления

Код для запуска

```
1: SELECT databasepropertyex('SQLSpyNet', 'RECOVERY')
```

Переход к другой модели восстановления

Вы можете перейти к другой модели восстановления, однако перед этим и после этого нужно кое-что сделать.

- От полной к массовой. При таком переходе менять что-либо в процедурах резервного копирования не нужно.
- От полной к простой. Необходимо скопировать журнал непосредственно перед переходом. Это обеспечит восстановление базы данных на момент перехода. Когда переход завершен, прекратите протоколирование в журнал.
- От массовой к полной. Если требуется восстановление на любой момент времени, следует немедленно выполнить резервное копирование журнала.
- От массовой к простой. Необходимо выполнить те же действия, что и при переходе от полной модели к простой.
- От простой к полной. Требуется выполнить резервное копирование базы данных сразу после перехода, а затем регулярно копировать базу данных и журнал.
- От простой к массовой. Необходимо выполнить те же действия, что и при переходе от простой модели к полной.

Однако в большинстве случаев после установки модели восстановления базы данных вам не придется переходить к другой модели. Исключение составляет случай, когда обнаруживается, что полная модель слишком быстро заполняет дисковое пространство или процессами протоколирования существенно затрудняет выполнение транзакций.

Для замены используемой модели восстановления необходимо выполнить оператор ALTER DATABASE, аналогично приведенному в листинге 10.2.

Листинг 10.2. Изменение модели восстановления

Код для запуска

```
1: ALTER DATABASE SQLSpyNet SET RECOVERY FULL
```

Этот оператор устанавливает для базы данных SQLSpyNet полную модель восстановления.

Выбор времени резервного копирования

Все это хорошо, но когда же следует создавать резервные копии базы данных? Вот вопрос ценой в миллион долларов! Причем в большей степени с точки зрения пользователей, а не из-за влияния на систему.

SQL Server 2000 позволяет создавать резервные копии, когда пользователи подключены. Это значит, что в процессе резервного копирования нормальная работа базы данных может не прекращаться. Однако при выполнении резервного копирования в таком режиме следует учесть некоторые обстоятельства.

- Оператор ALTER DATABASE с параметрами ADD FILE или REMOVE FILE остановит процесс резервного копирования.
- В процессе копирования операторы INSERT, UPDATE и DELETE могут выполняться без ограничений.
- Сокращение базы данных или сокращение файла повлекут крах копирования. Значение этих терминов рассматривается в главе 14, "Отладка и устранение ошибок в SQL Server 2000".

Операция резервного копирования может занять довольно много времени (в зависимости от размера базы данных), поэтому планируйте ее на спокойное время. В системе, работающей 24 часа в день и 7 дней в неделю, нужно найти время, когда она загружена меньше всего (обычно это между 2.00 и 3.00), и выполнить в это время резервное копирование.

Как, для этого я должен выходить на работу в два часа ночи? Вы шутите! Однако успокойтесь. Вскоре вы узнаете, как составить расписание автоматического выполнения этого процесса в заданное время.

Где хранить резервные копии

Их можно хранить в разных местах, SQL Server 2000 позволяет создавать их там, где мы ему укажем.

- На диске. Можно создать резервную копию базы данных как один файл (обычно с расширением .bak) и поместить его на один из наших дисков. При этом реко-

мендуется придерживаться соглашения об именах, например: `C:\Program Files\Microsot SQL Server\MSSQL$MYSQLSERVER\BACKUP\SQLSpyNet.bak`.

- На ленте. Можно поместить резервную копию нашей базы данных на ленточный накопитель. Для него тоже необходимо указать полный маршрут, например: `\\.\TAPE1`.

Поскольку базу данных можно скопировать непосредственно на диск, мы можем указать серверу место, куда необходимо копировать, и содержать таким образом резервные копии вне сервера. Если резервное копирование выполняется на ленточный накопитель, то копии ленты следует хранить в другом помещении и желательно в прочном сейфе. Лучше перестраховаться, чем потом жалеть.

Что именно следует копировать

Это один из самых изменчивых аспектов резервного копирования. Кроме типа модели восстановления, дополнительно можно задавать, какие части базы данных мы хотим включать в резервные копии.



Прежде чем выбрать тип резервного копирования, обычно требуется проанализировать модель восстановления базы данных.

SQL Server 2000 поддерживает описанные ниже опции резервного копирования.

- Database - complete. Полное копирование. База данных полностью копируется в ее текущем состоянии. Если установлен этот переключатель, копируются как данные, так и все объекты (таблицы, представления, хранимые процедуры и т.д.). С помощью такой резервной копии базу данных можно восстановить на другом сервере.
- Database - differential. Частичное копирование. Копируются только изменения происшедшие со времени последнего полного копирования базы данных. Поэтому если была добавлена новая таблица, то она будет включена в резервную копию. Аналогично, в резервную копию включаются все изменения данных.
- Transaction log. Копируется журнал транзакций (т.е. все транзакции пользователей, включая операторы `INSERT`, `UPDATE` и `DELETE`). Это позволяет восстановить базу данных на любой момент времени. Поскольку резервная копия содержит все транзакции пользователей, могут быть восстановлены все введенные ими данные.
- File and filegroup. Осуществляется копирование некоторой части базы данных за один раз. Эту опцию следует использовать только для огромных баз данных, для которых процесс резервного копирования занимает очень много времени. Чтобы можно было восстановить базу данных, необходимо копировать также журнал.

Теперь вы подготовлены к резервному копированию теоретически. Перейдем к практике. Как и все в SQL Server 2000, резервное копирование может быть выполнено двумя способами. Это можно сделать в Enterprise Manager с помощью графических инструментов или путем выполнения операторов Transact-SQL в окне Query Analyzer. Как вы понимаете, я предпочитаю второй метод.

Однако, чтобы вы были всесторонне подготовленным администратором баз данных, рассмотрим оба метода.



На заметку

Исходя из собственного опыта, могу отметить, что при использовании графических инструментов процесс резервного копирования подвержен ошибкам более других. В то же время, если ошибка происходит при выполнении сценария резервного копирования, то достаточно исправить ее в сценарии, и резервное копирование будет успешно завершено.

Резервное копирование базы данныхSQLSpyNet с помощью *Enterprise Manager*

Итак, у нас есть ценная база данных, которую необходимо защитить. Мы хотим выполнить резервное копирование, которое позволит легко восстановить базу данных. Транзакции наших пользователей невелики, поэтому мы можем протоколировать каждое событие (полная модель восстановления). Это наше первое резервное копирование, поэтому выполним его с использованием параметра Database - complete (полное копирование базы данных).

1. Запустите Enterprise Manager и найдите базу данных SQLSpyNet. Щелкните правой кнопкой мыши на базе данных и выберите команду All Tasks⇨Backup Database (рис. 10.1).

Рис. 10.1. Команды резервного копирования базы данных SQLSpyNet

При выборе этой команды появляется окно, показанное на рис. 10.2.

2. В этом окне представлено несколько параметров резервного копирования базы данных. Первый параметр — имя копируемой базы данных. Введите **SQLSpyNet**.

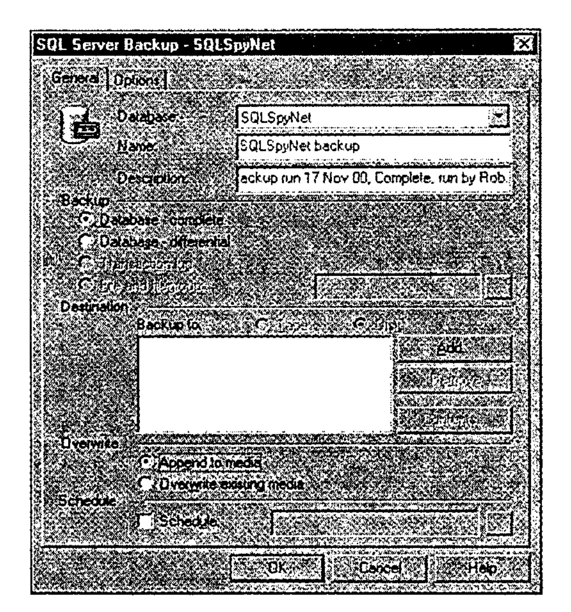


Рис. 10.2. Окно для установки параметров резервного копирования базы данных SQLSpyNet

3. Далее следует выбрать логическое имя резервной копии базы данных (Name). Оно должно что-то означать для нас, чтобы резервную копию можно было легко узнать. Эта опция обязательна. Введите в этом поле **SQLSpyNet backup**.
4. Теперь введем описание резервной копии базы данных. В поле Description введите **Полная резервная копия базы данных SQLSpyNet; сделана *дата*; выполнил *имя***.
5. Далее необходимо выбрать тип резервной копии. Установите переключатель Database - complete.

На заметку

Вы наверняка заметили, что переключатели *Transaction log* и *File and filegroups* недоступны. Переключатель *Transaction log* недоступен потому, что при создании базы данных активизирован параметр *Truncate log on checkpoint* (Усекать журнал в точках сохранения).

Поэтому, если наша база данных потерпит крах, мы сможем восстановить ее только на момент последнего копирования, а не на любой момент времени. В коммерческих базах данных использовать эту опцию не рекомендуется, потому что при крахе базы данных и последующем восстановлении вам придется просить пользователей повторно ввести все данные, введенные с момента последнего копирования. Поверьте мне, это им очень не понравится.

Мы используем этот параметр лишь потому, что пока только разрабатываем базу данных. Другими словами, данные не представляют для нас большой ценности, в случае краха их легко ввести повторно.

Изменить описываемый параметр можно двумя способами (вам это уже знакомо). Первый — использование оператора Transact-SQL:

```
sp_dboption 'SQLSpyNet', 'trunc. log on chkpt', 'true'
```

Второй — использование окна свойств базы данных (более подробно эта операция описана в главе 3, “Вербовка “виртуальных” агентов, или Создание базы данных SQLSpyNet”).

6. В разделе Destination (Назначение) следует задать место, где мы хотим хранить резервную копию. Щелкните на кнопке Add. Появится диалоговое окно, в котором можно ввести путь к файлу с резервной копией (рис. 10.3).

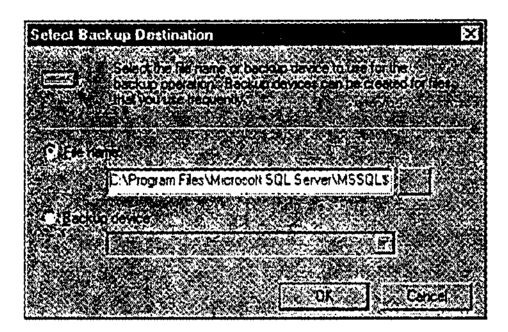


Рис. 10.3. Определение пути к файлу резервной копии базы данных SQLSpyNet

7. Введите путь с полным именем файла, например **C:\Program files\Microsoft SQL Server\MSSQL$MYSQLSERVER\BACKUP\SQLSpyNetEM.bak**.

   Путь также можно указать с помощью кнопки с троеточием (...). При щелчке на ней появляется окно, позволяющее передвигаться по файловой системе. Найдя место, где вы хотите сохранить файл резервной копии базы данных, введите его имя в поле File name (рис. 10.4).

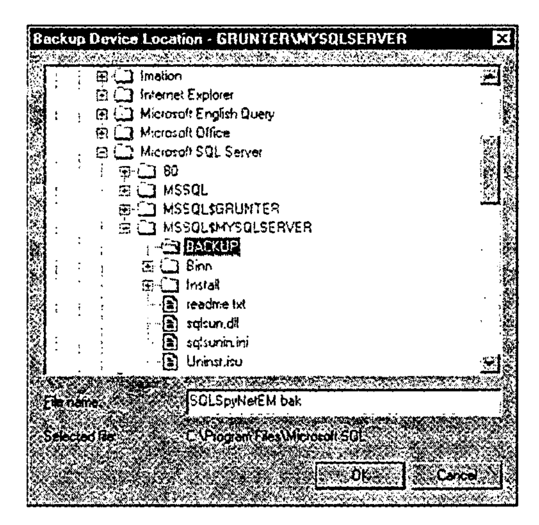


Рис. 10.4. Альтернативный способ указания места хранения резервной копии базы данных SQLSpyNet

8. Щелкайте на кнопках OK, пока не вернетесь в первое окно. В нем вы увидите запись в поле Backup to. Рассмотрим кнопки и параметры, расположенные рядом с этим полем.

   - С помощью кнопки Remove можно удалить любую копию из списка имеющихся резервных копий.

- С помощью кнопки Contents можно увидеть текущее число резервных копий. Например, если выполнить резервное копирование базы данных сегодня, завтра и послезавтра, то после этого в поле содержимого вы увидите три записи с датами и временем создания резервных копий, размерами файлов и т.д.
- Переключатель Overwrite existing media (Переписать текущий носитель) устанавливает запись резервной копии вместо предыдущей копии. Если этот переключатель установлен, то вы не сможете вернуться к прежним состояниям базы данных. Это значит, что если вы выполнили одно резервное копирование сегодня, а второе завтра, то восстановить вчерашнее состояние базы данных завтра будет уже невозможно. Сейчас оставьте установленный по умолчанию переключатель Append to media (Присоединить).
- Флажок Schedule (Расписание). В свое время вас очень встревожила перспектива подниматься среди ночи для выполнения резервного копирования, но можете не волноваться, наши друзья в Microsoft позаботились об этом.
- С помощью параметра Schedule можно создать расписание автоматического выполнения резервного копирования. Это похоже на систему напоминания в Outlook. Мы можем таким образом обеспечить периодическое выполнение необходимой задачи. Сейчас, однако, оставьте этот флажок снятым.

9. Вы, видимо, заметили в этом окне еще одну вкладку. Во вкладке Options находятся дополнительные параметры резервного копирования.

   - Флажок Verify backup upon completion (Проверка копии при завершении) открывает окно сообщений, информирующее об успешной проверке целостности данных и завершении копирования. Установите его.
   - Флажок Check media set name and backup set expiration (Проверка установленных имен носителей и времени существования резервных копий) включает проверку параметров носителя таким образом, чтобы не были переписаны существующие копии, которые могут еще понадобиться. Назначение этого параметра подробно изложено в справочной системе Books Online.

     Можно задать имя носителя, на который выполнятся резервное копирование. При каждом выполнении сервером SQL Server 2000 резервного копирования на носитель, можно получить его имя для проверки различных параметров. Снимите этот флажок.

10. Щелкните на кнопке OK, и SQL Server 2000 приступит к выполнению резервного копирования. На экране появится окно, показанное на рис. 10.5.

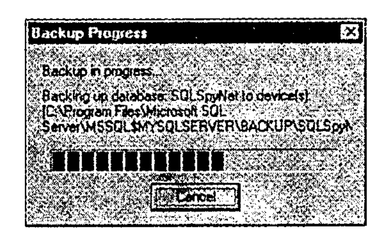


Рис. 10.5. SQL Server 2000 выполняет резервное копирование

Время выполнения резервного копирования зависит от параметров компьютера, размера базы данных, быстродействия жестких дисков и некоторых других факторов. Однако наша база данных небольшая, и это не должно занять много времени даже на моем Pentium 75! Когда SQL Server 2000 завершает резервное копирование, появляется окно, информирующее о том, что операции проверки целостности и резервного копирования завершены успешно.

Итак, дамы и господа, мы успешно выполнили наше первое резервное копирование в SQL Server 2000. Согласитесь, это было нетрудно.

На заметку Теперь посмотрите в файловой системе каталог, выбранный для хранения резервной копии. В нем должен быть файл резервной копии с расширением `.bak`.

Резервное копирование базы данных SQLSpyNet с помощью операторов Transact-SQL в окне *Query Analyzer*

Как видите, создать резервную копию с помощью Enterprise Manager довольно просто, однако большое число действий делает эту задачу весьма трудоемкой. Сейчас мы выполним точно такое же резервное копирование, но на этот раз с помощью кода Transact-SQL. Так что открывайте окно Query Analyzer и — вперед!

В листинге 10.3 приведен код Transact-SQL, выполняющий резервное копирование.

Листинг 10.3. Резервное копирование базы данных SQLSpyNet

Код для запуска

```
1: BACKUP DATABASE SQLSpyNet
1a: TO DISK = 'C:\Program Files\Microsoft SQL Server\
    ➥MSSQL$MYSQLSERVER\BACKUP\SQLSpyNetQA.bak'
1b: WITH NOINIT, NOSKIP, STATS = 10
```

Этот сценарий выполняет резервное копирование базы данных и выводит на экран сообщения о текущем состоянии процесса резервного копирования (рис. 10.6).

Рассмотрим оператор `BACKUP DATABASE`.

Анализ

- Строка 1. Сообщает SQL Server 2000, что нужно выполнить резервное копирование базы данных SQLSpyNet.
- Строка 1a. Параметр `TO DISK` определяет путь и полное имя файла создаваемой резервной копии.
- Строка 1b. Это необязательная часть оператора. Однако с помощью этих параметров выполняются те же действия, что и при выборе некоторых элементов управления Enterprise Manager. Слово `WITH` означает, что за ним последуют параметры. Параметр `NOINIT` означает, что набор резервной копии добавляется в старую копию, а не записывается вместо нее. Эта опция аналогична снятию флажка Overwrite existing media. Если в Enterprise Manager он снят, то резервная копия добавляется, а не переписывается вместо старой. Параметр `NOSKIP` определяет проверку имени и параметров носителя. Аналогично флажку Check media set name and backup set expiration в Enterprise Manager, сервер проверяет время жизни (если оно есть) наборов резервной копии, прежде чем удалить их и записать новые. И наконец, параметр `STATS = 10` сообщает SQL Server 2000, что в процессе резервного копирования он должен докладывать о текущем состоянии выполнения задачи, например: "скопировано 10%", "скопировано 20%" и т.д.

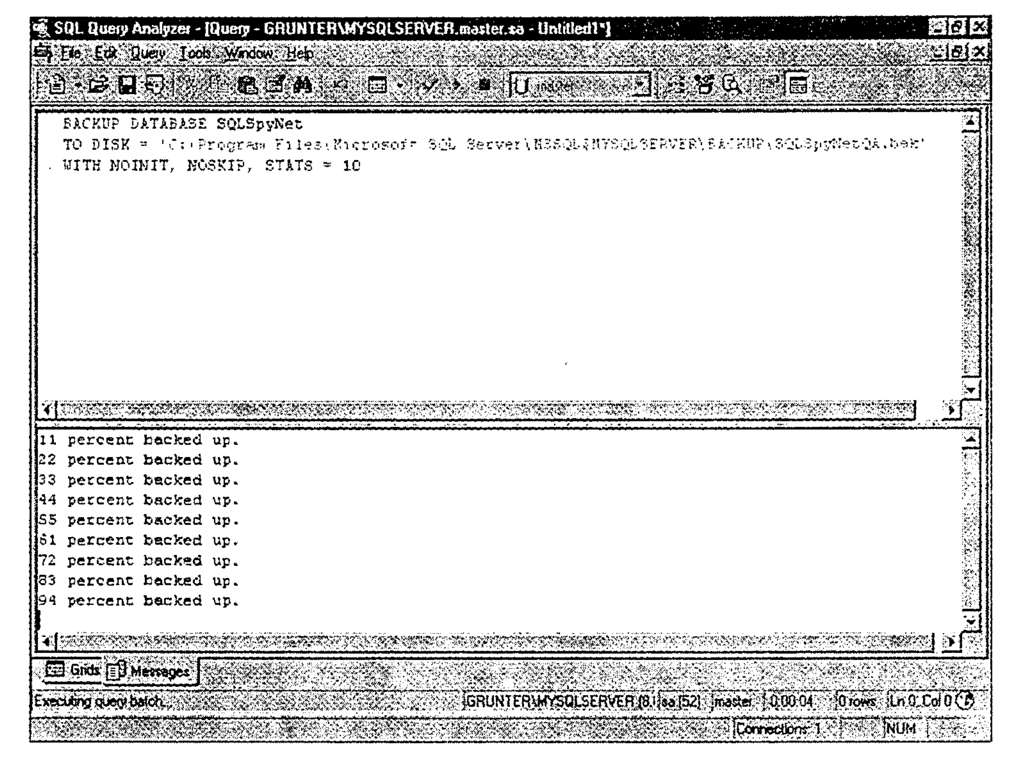


Рис. 10.6. SQL Server 2000 выполняет второе резервное копирование базы данных

На заметку Если установить параметр `STATS = 10`, то шаг состояния о выполнении не обязательно будет равен точно 10%. Иногда сервер может сообщить о 8%, затем о 19% и т.д.

Теперь в каталоге резервного копирования вы увидите не один, а два файла резервной копии. Один из них помечен как EM (Enterprise Manager), а второй как QA (Query Analyzer), если, конечно, вы придерживались всех требований.

Итак, у нас есть два файла с резервными копиями. Теперь вы можете решить, какой метод резервного копирования лучше!

Восстановление базы данных SQLSpyNet

Пока мы выполнили только половину плана резервного копирования. Теперь рассмотрим, как использовать только что созданные резервные копии для восстановления базы данных. Когда все испортится (а это непременно когда-нибудь произойдет), эти резервные копии вам очень пригодятся.

У нас есть только резервные копии базы данных и журнала транзакций, поэтому мы не можем гарантировать восстановление базы данных непосредственно в то состояние, которое она имела в момент аварии.

Когда мы выполняем резервное копирование базы данных, то копируем ее текущее состояние. При восстановлении базы данных мы восстанавливаем ее состояние на момент копирования. Если в резервной копии есть незавершенные транзакции, то при восстановлении они отменяются. Этим обеспечивается целостность данных.

Незавершенные обновления базе данных ни к чему, не так ли? Как вы помните, транзакции должны удовлетворять требованиям ACID (см. главу 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок").

Если при выполнении резервного копирования пользователь что-то делал (еще один довод в пользу ночного копирования) и после этого произошел крах системы, то все данные этого пользователя будут утеряны. Таким образом, хотя резервное копирование и помогает восстановить данные, сделать это на все 100% невозможно.

Как получить все обратно

Итак, вы выполнили резервное копирование базы данных. В этот момент прибегает взволнованный пользователь и сообщает, что он совершенно случайно удалил всех агентов из таблицы Spy, и, предупреждая ваш вопрос, говорит, что не использовал транзакции.

После 10-минутной нотации о важности транзакций, вы идете искать последние резервные копии и видите, что на диске их более 20. Какую из них взять?

К счастью для нас, SQL Server 2000 содержит всю информацию о резервных копиях в базе данных msdb. Поэтому, если резервное копирование выполнялось каждую ночь, вся информация об этом хранится в msdb. На основе информации в msdb SQL Server 2000 предлагает параметры восстановления базы данных. Это значительно облегчает поставленную задачу. Только представьте, насколько труднее было бы рыться среди 20 старых файлов с резервными копиями, отыскивая среди них наиболее подходящий.

С помощью утилиты восстановления можно просмотреть имена и описания резервных копий, что очень поможет в поиске нужной копии.

Восстановление базы данных

Что происходит при восстановлении базы данных? Если восстанавливаемой базы данных на сервере не существует, то SQL Server 2000 создает ее. Например, если жесткий диск потерпел крах и требуется заново устанавливать сервер, то база данных, естественно, тоже пропала. Таким образом, SQL Server 2000 создает заново базу данных с теми же объектами и параметрами, которые существовали на момент последнего резервного копирования. Вам при этом не придется заново создавать все объекты и настраивать параметры базы данных.



Это, конечно, не означает, что если база данных существует, то произойдет ошибка. В этом случае SQL Server 2000 перепишет текущую базу данных, заменив все ее содержимое тем, что было сохранено в резервной копии.

В предыдущих версиях SQL Server для восстановления базы данных нужен был сервер с теми же параметрами сортировки, которые установлены на сервере, выполняющем резервное копирование (см. приложение Б, "Установка и настройка SQL Server 2000"). Ни больше, ни меньше! А как же быть, если этого сервера уже нет? К счастью, SQL Server 2000 поддерживает изменение параметров сортировки без повторной переустановки.

Можно также восстановить базу данных, работавшую под управлением Windows 98 на компьютере с Windows NT/2000 и наоборот. Если нашей базе данных SQLSpyNet суждена долгая жизнь, то, скорее всего, мы перенесем ее на сервер под управлением Windows 2000.

Еще одна весьма приятная новость — автоматическая настройка на более высокую версию при восстановлении базы данных SQL Server 7.0. Если вы создали резервную копию базы данных в SQL Server 7.0 по изложенной выше методике, то при ее восстановлении в SQL Server 2000 вам не придется делать абсолютно ничего! Переход на более высокую версию, таким образом, значительно облегчается.



Без проблем выполняется переход с версии 7.0 на 2000, однако более ранние версии SQL Server имели другую систему внутреннего хранения. Это значит, что для перехода, например, с SQL Server 6.5 понадобится помощь специального мастера обновления. Более подробно эта тема рассматривается в справочной системе Books Online.

Рассматривая систему резервного копирования, мы увидели, что можно создавать копии не только всей базы данных, но и ее частей, например журналов, групп файлов и т.д. Как можно использовать эти копии?

Применение журналов транзакций

Зачем может понадобиться копия журналов транзакций? Если полное копирование базы данных выполняется каждую ночь, а журналов транзакций — каждый час, то можно восстановить базу данных на момент последнего копирования журнала.

Посмотрим, что будет, если не копировать журналы вообще. Допустим, резервное копирование базы данных выполняется ежедневно в 3.00 (вы в это время, видимо, сладко спите). Ваши пользователи приходят в 8.00 и усиленно загружают базу данных вплоть до 2.00 ночи. Следующее резервное копирование будет через час, однако в этот момент диск терпит крах! Страшная катастрофа! Из сервера поднимается легкий дымок и вам кажется, что из корпуса компьютера выглядывают странные обломки.

Как настоящий администратор баз данных, вы приносите другой компьютер, запускаете на нем другой сервер и восстанавливаете последнюю резервную копию. Однако она устарела почти на 24 часа! Тяжелая работа, проделанная за целые сутки, оказалась безвозвратно утерянной. И вам предстоит сообщить это усталым и взволнованным людям, собравшимся вокруг вас в ожидании новостей!

Чем в этом случае помогут резервные копии журнала? Если у вас есть полная копия базы данных с последней ночи и, кроме того, ежечасные резервные копии журнала, то с их помощью можно восстановить базу данных на момент последнего копирования журнала (а если повезет и после этого изменений данных не было, то и на момент краха). Другими словами, в худшем случае пользователи потеряют только час работы. Это намного лучше, чем 23 часа, не так ли?



С нашими текущими установками базы данных мы не можем выполнять резервное копирование журнала. Дело в том, что у нас, как вы помните, активизирован параметр *Truncate log on checkpoint*. Он предотвращает непрерывный рост журнала, однако в коммерческих базах данных активизировать его не рекомендуется по только что изложенным причинам.

В нашем примере не требуется высокий уровень восстановления данных, поэтому параметр *Truncate log on checkpoint* у нас активизирован. Пока нам это подходит больше, потому что непрерывно растущий журнал транзакций может быстро заполнить наш небольшой диск.

Для восстановления базы данных с помощью резервных копий журнала транзакций необходимо придерживаться ряда правил.

- Сначала должна быть восстановлена база данных с полной или частичной резервной копии.
- Должна существовать непрерывная последовательность резервных копий журнала с момента последнего полного или частичного копирования базы данных.
- Журналы транзакций должны быть расположены в правильной последовательности.
- После восстановления базы данных никакая транзакция больше не может быть применена (если только вы не собираетесь повторить весь процесс сначала).

Как и резервное копирование, восстановление базы данных можно выполнить двумя разными способами: с помощью графических инструментов Enterprise Manager или путем выполнения кода Transact-SQL в окне Query Analyzer. Рассмотрим оба метода восстановления.

На заметку Мы не будем выполнять резервное копирование журнала или восстановление с его помощью. Если вы хотите попробовать, то найдите в Books Online операторы Transact-SQL `BACKUP LOG` и `RESTORE LOG`.

Восстановление SQLSpyNet с помощью *Enterprise Manager*

Как и в случае резервного копирования, все необходимые действия можно выполнить с помощью Enterprise Manager.

1. Откройте окно Enterprise Manager, найдите папку базы данных SQLSpyNet, щелкните на ней правой кнопкой мыши и из контекстного меню выберите команду All Tasks⇨Restore Database. Появится окно, показанное на рис. 10.7.

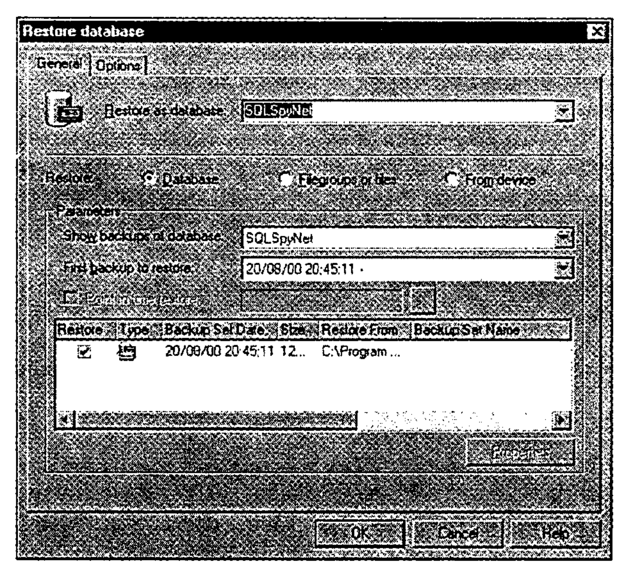


Рис. 10.7. Окно восстановления базы данных SQLSpyNet в SQL Server 2000

Вкладка General содержит много параметров, что делает процесс восстановления довольно гибким.

2. Сначала нужно выбрать базу данных, подлежащую восстановлению. Если она существует, выберите ее из раскрывающегося списка. Если нет, то введите имя новой базы данных и время восстановления, в этом случае SQL Server 2000 создаст ее. Ну а сейчас выберите из раскрывающегося списка SQLSpyNet.
3. В разделе Restore можно выбрать тип восстановления. В этом разделе доступны три параметра.
   - Database. Восстановление выбранной базы данных. В расположенном ниже поле со списком приведена информация о последних резервных копиях базы данных. Эта информация хранится в базе данных msdb.

     Мы собираемся восстанавливать последнюю копию, поэтому установите переключатель Database.
   - Filegroups or files. Восстановление файла или группы файлов, т.е. восстанавливается подмножество полной базы данных.
   - From device. Восстановление базы данных из файла, хранящегося на ленте или на другом диске. Этот параметр используется, когда в приведенном ниже списке нет нужной резервной копии.

На заметку

Остальные параметры пока оставим.

4. Выберем базу данных, для которой мы хотим увидеть список резервных копий. Убедитесь, что выбрана SQLSpyNet.
5. В раскрывающемся списке First backup to restore (Первая восстанавливаемая база данных) приведен перечень последних резервных копий в обратной последовательности, т.е. последняя резервная копия расположена в нем первой. Выберите в списке последнюю резервную копию. Если выбрать другую резервную копию, то информация о ней будет добавлена в поле со списком. Таким образом можно восстанавливать несколько резервных копий одну за другой.
6. Если установить флажок в поле Restore, то становится доступной кнопка Properties. С ее помощью можно редактировать свойства восстановления, например изменить файл, из которого будет восстановлена база данных.
7. Как и окно резервного копирования, окно восстановления имеет еще одну вкладку — Options. В ней можно изменить некоторые (перечисленные ниже) параметры восстановления резервной копии.

На заметку

Оставьте неизменными параметры этой вкладки.

- Eject tapes (if any) after restoring each backup (Извлечь ленту (если она есть) после восстановления каждой копии). При завершении восстановления лента извлекается из накопителя.
- Prompt before restoring each backup (Подтверждать восстановление каждой резервной копии). Если установить этот флажок, то SQL Server 2000 перед восстановлением каждой копии будет спрашивать, продолжить ли восстановление.

- Force restore over existing database (Восстановив, записать поверх существующей базы данных). Существующая база данных удаляется, и во вновь созданную заносится информация, хранящаяся в резервной копии.
- Restore database files as (Восстановить эти файлы базы данных). Если нужно восстановить другие файлы базы данных, то здесь следует ввести их имена.

8. Убедившись, что все параметры вкладки General установлены верно, щелкните на кнопке OK. Появится окно, показанное на рис. 10.8.

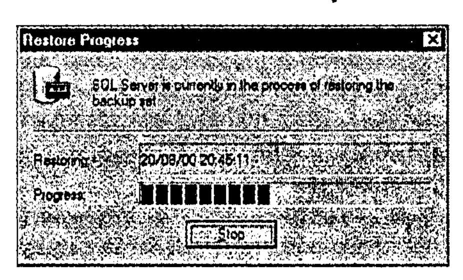


Рис. 10.8. Протекает процесс восстановления базы данных SQLSpyNet

Дело сделано! Теперь, если все нормально, осталось только дождаться сообщения SQL Server 2000 об успешном завершении восстановления.

Мы рассмотрели восстановление базы данных с помощью графических средств Enterprise Manager. Далее рассмотрим восстановление с помощью кода Transact-SQL в окне Query Analyzer.

Восстановление базы данных SQLSpyNet с помощью операторов Transact-SQL

Запустите Query Analyzer и введите код Transact-SQL, приведенный в листинге 10.4.

Листинг 10.4. Восстановление базы данных Spy Net с помощью кода Transact-SQL

Код для запуска

```
1: RESTORE DATABASE SQLSpyNet
2: FROM DISK = 'C:\Program Files\Microsoft SQL Server\
   ↳MSSQL$MYSQLSERVER\BACKUP\SQLSpyNetQA.bak'
3: WITH STATS = 10
```



Если оператор RESTORE терпит крах, посмотрите, не активна ли база данных *SQLSpyNet* в окне *Enterprise Manager*. Если эта база данных уже выбрана, выберите другую, например master. Тогда база данных не будет иметь установленных соединений и ее восстановление может быть продолжено.

Строки листинга 10.4 имеют следующее назначение.

Анализ

- Строка 1. Сообщает SQL Server 2000, что мы хотим восстановить базу данных SQLSpyNet.
- Строка 2. Определяет место хранения восстанавливаемой резервной копии. В нашем примере она хранится на диске.
- Строка 3. Указывает вывод на экран отчета о состоянии процесса восстановления.

Итак, мы опять восстановили нашу базу данных, на этот раз с помощью кода Transact-SQL (ура!). Операторы Transact-SQL выполнили для нас те же действия, что и средство Enterprise Manager.

Завершение плана резервного копирования и восстановления

Что еще нужно знать о параметрах резервного копирования и восстановления? Прежде всего при разработке плана восстановления нужно оценить материальные потери, вызванные потерей данных. Это позволит выбрать оптимальный в данном случае тип резервного копирования и восстановления. План восстановления должен быть хорошо продуман и обеспечивать как максимальную защиту данных, так и возможность быстрого восстановления.

При составлении плана нужно также учесть требования безопасности данных, наличие доступных ресурсов, необходимость обеспечения доступности данных. Нужно также составить таблицу контрольных проверок. В данном разделе приведены некоторые полезные советы и подсказки на этот счет.

Следите за вашими копиями

В главе 9, "Обеспечение безопасности базы данных Spy Net", описано, как предоставить или отменить право пользователя на выполнение резервного копирования и восстановления. SQL Server 2000 предоставляет дополнительное средство защиты резервных копий — пароли. Теперь, если кто-нибудь даже раздобудет вашу резервную копию, он не сможет ее восстановить.

Однако Microsoft утверждает, что пароли — лишь базовый уровень безопасности. Пароль не шифрует ни данных, ни наборов резервных копий. Его можно взломать. Но ведь это только одно из средств! Это как замок на двери вашего дома. Его тоже можно взломать, тем не менее это довольно полезное и необходимое средство защиты вашего имущества.

Выполнять резервное копирование и восстановление можно также с помощью мастера копирования базы данных. При его использовании дополнительно копируются учетные записи пользователей. Если база данных содержит задания с расписаниями (см. главу 11, "Администрирование разведывательной сети"), то мастер скопирует их тоже. Мастер копирования базы данных рассматривается в главе 15, "Самостоятельное исследование SQL Server 2000".

Помечайте журналы транзакций

В журнале транзакций можно пометить определенное место аналогично закладке в книге. Если база данных окажется поврежденной неудачным обновлением, то закладка поможет найти в журнале транзакций место, расположенное непосредственно перед этим повреждающим обновлением.

Например, если выполняется обновление нескольких тысяч строк, то полезно отметить в журнале точку непосредственно перед этим обновлением. Если после обновления окажется, что оно выполнено неправильно, можно будет повторно выполнить транзакции до отметки в журнале. База данных таким образом будет восстановлена на момент, непосредственно предшествующий повреждающему обновлению.

На заметку Сделать отметку в журнале транзакций можно с помощью оператора `BEGIN TRANSACTION WITH MARK` *имя_транзакции*, который вставляет в него пометку. (Транзакции рассматривались в главе 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок".) Восстановить базу данных до отмеченной транзакции можно с помощью оператора `RESTORE` с директивой `WITH STOPMARK =` *имя_транзакции*.

Еще одна новинка — "теплый" сервер

Для защиты данных можно воспользоваться еще одним новым средством, которое называется *переносом журнала*. С его помощью можно постоянно создавать резервные копии журналов базы данных и восстанавливать их на другом сервере. Таким образом, постоянно будет существовать резервный, так называемый теплый, сервер, практически полностью идентичный рабочему серверу.

Термин *Теплый сервер* имитирует все действия основного сервера, ожидая его выхода из строя. Когда это произойдет, достаточно отсоединить от сети основной сервер и присоединить теплый. Нормальная работа при этом будет возобновлена в течение минуты.

Кроме рассмотренных, SQL Server 2000 предоставляет еще довольно много средств резервного копирования и восстановления, которые будут описаны позже. Сейчас ограничимся только этими, чтобы обострить ваш аппетит.

Создание списка контрольных проверок ресурсов и процессов

В приведенный ниже список я включил некоторые ресурсы, которые вы должны учитывать при создании плана. Этот список далеко не полный, он даст вам только общее представление того, о чем вы должны позаботиться при создании плана аварийного восстановления. Составьте собственный список и держите его под рукой, он в любой момент может понадобиться вам или дежурному на сервере.

- Назначьте ответственного за восстановление. Другими словами, человека, который обеспечивает постоянную готовность и высокое качество плана аварийного восстановления.
- Регулярно создавайте резервные копии баз данных. Это относится не только к пользовательским, но и к системным базам данных.
- Отслеживайте все изменения сервера. Сюда входят программное обеспечение, служебные пакеты, обновления и все прочее, что может быть установлено на сервере или серверах.
- Выполняйте контрольные восстановления. В соответствии с планом восстановите базы данных на другом сервере. Это поможет убедиться в работоспособности и качестве плана аварийного восстановления.
- Подготовьте теплый сервер. Если это возможно, то такая мера значительно повысит надежность всей системы.
- Позаботьтесь о доступности поддержки. В аварийной ситуации могут возникнуть непредвиденные обстоятельства, с которыми вы не сможете справиться самостоятельно, несмотря на хороший план. Поэтому не отказывайтесь от контакта с людьми и организациями, способными оказать поддержку.

Создайте план восстановления и придерживайтесь его

Что делать, когда сервер все же терпит крах? Только без паники! Медленно и методично следуйте вашему плану аварийного восстановления вплоть до мельчайших деталей. Именно для этого вы затратили много часов на его создание.

Вот один из возможных планов аварийного восстановления.

1. Переустановите сервер заново, применяя программное обеспечение, сервисные пакеты, обновления и все остальное, что есть в вашем списке программного обеспечения.
2. Когда SQL Server 2000 на вашем сервере восстановлен, в первую очередь восстановите системные базы данных. Обычно для этого достаточно перезагрузить сервер.
3. Восстановите пользовательские базы данных.
4. Установите соединение сервера с клиентскими приложениями.
5. Протестируйте приложение.
6. Если предыдущие шаги выполнены успешно, предоставьте пользователям доступ к системе.
7. Следующие 24 часа внимательно анализируйте производительность и характер выполнения процессов на сервере.

Примерно так выглядит короткий план аварийного восстановления. Однако запомните: планируйте свои действия и делайте это, не откладывая! Если вы отложите создание плана на потом, то согласно закону Мерфи аварийная ситуация возникнет именно тогда, когда у вас еще не будет плана действий.

Резюме

В этой главе рассмотрена главная обязанность администратора баз данных: обеспечить быстрое восстановление работоспособности системы после самой тяжелой аварии. Описаны основы применения операторов резервного копирования и восстановления. В то же время многие аспекты данной проблемы остались не затронутыми. Настоятельно рекомендую уделить этой задаче больше внимания при самостоятельном изучении, прежде чем приступать к работе с коммерческой системой.

Хотя мы и рассмотрели планирование действий в аварийной ситуации, но фактически не определили правила создания плана. В значительной степени план зависит от реальных потребностей пользователей, руководства и разработчиков. Настоятельно рекомендую планировать стратегию резервного копирования и восстановления *очень* тщательно. Разработанный план нужно тестировать, тестировать и еще раз тестировать!

К сожалению, мы не сможем реально оценить сильные и слабые стороны плана, пока он нам не понадобится. Однако Microsoft предоставила неплохие материалы по разработке плана резервного копирования и восстановления. Их можно найти в справочной системе Books Online.

И еще одно замечание относительно управления резервными копиями. Вы должны четко представлять себе, когда, кому и зачем могут понадобиться ваши резервные копии. В Books Online можно найти очень хорошие материалы также и по этому вопросу.

Следующие шаги

В следующей главе рассматривается создание расписаний автоматического выполнения резервного копирования. Расписание позволит выполнять копирование в самое удобное ночное время, при этом не придется приходить на работу в три часа ночи только для того, чтобы щелкнуть на кнопке Start. Как видите, Microsoft не пожалела усилий для нашего с вами удобства.

Глава 11

Администрирование разведывательной сети

В этой главе...

Теперь приступаем к рутинным, но жизненно важным задачам администрирования базы данных. В этой главе рассматриваются реальные задачи, выполняемые администратором базы данных, так что приготовьтесь побыть на этой должности еще некоторое время.

В предыдущих главах было описано, как спроектировать базу данных, как реализовать этот проект и как выполнять резервное копирование и восстановление базы данных. Теперь рассмотрим, как с помощью встроенных инструментов и различных средств SQL Server поддерживать работоспособность базы данных на достаточно высоком уровне.

В этой главе начинается обсуждение некоторых важных задач администрирования базы данных. Как администратору базы данных, вам часто придется выполнять эти задачи, причем настолько часто, что захочется узнать о них как можно больше; и, поверьте мне, только на тему администрирования написаны десятки книг.

Как упоминалось ранее (в главе 2, "Компоненты SQL Server 2000"), SQL Server 2000 предоставляет богатый набор инструментов для выполнения самых разнообразных задач, в том числе и задач администрирования. Графический интерфейс пользователя значительно облегчает жизнь администратора, однако не освобождает от необходимости планирования и анализа работы баз данных.

Обязанность администратора — обеспечить не только доступность баз данных, но и высокую эффективность работы сервера. Это значит, что если сервер работает плохо, то виноваты в этом только мы.

Другая обязанность администратора баз данных — обеспечить пользователям системы доступ к серверу и базе данных, когда им это необходимо. Эта цель достигается путем правильного конфигурирования опций сервера.

Конфигурирование SQL Server 2000

При установке SQL Server 2000 многие опции определены по умолчанию. В большинстве случаев они не нуждаются в переустановке, но если SQL Server 2000 рабо-

тает неэффективно или медленно, то его работу можно улучшить, настроив на использование собственных ресурсов.

Возможности конфигурирования сервера хорошо детализированы. Можно даже указать, какой объем оперативной памяти должен использовать SQL Server. Если в Enterprise Manager щелкнуть правой кнопкой на имени сервера и выбрать команду Properties (Свойства), откроется диалоговое окно (рис. 11.1), в котором можно настраивать свойства сервера.

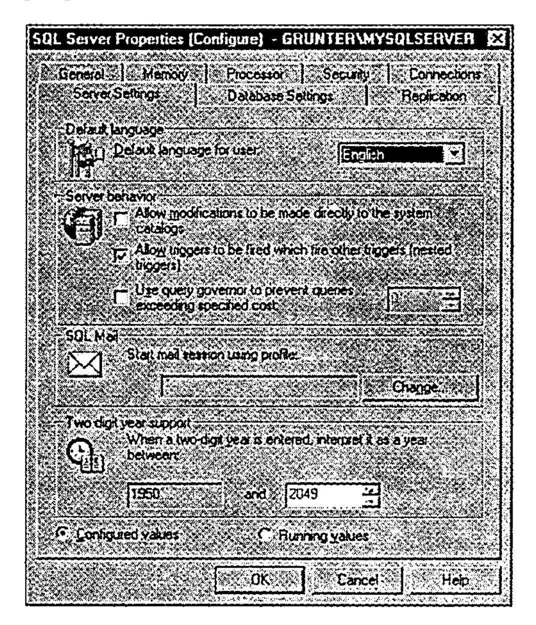


Рис. 11.1. Параметры конфигурации SQL Server 2000

Диалоговое окно свойств содержит довольно много параметров конфигурации. Большинство из них не нуждаются в изменении, однако следует знать, где найти самые необходимые.

Отметим наиболее интересные параметры.

- Во вкладке Server Settings можно задать для сервера (а следовательно, и для пользователей) язык по умолчанию.
- Параметр Allow modifications to be made directly to the system catalogs (Разрешить записывать изменения непосредственно в системный каталог) позволяет пользователям модифицировать системные таблицы.



Включайте эту опцию, только когда вы уверены в том, что это нужно. Если *системные каталоги* потеряют согласованность, база данных может породить серьезные ошибки или вообще оказаться бесполезной.



Системные каталоги состоят из системных таблиц баз данных. Системные таблицы содержат определения объектов базы данных, например имени объекта, столбцов (если они есть) и всех других свойств объектов. По умолчанию непосредственное изменение системных таблиц не разрешается. Вы можете позволить делать это, но будьте осторожны.

- Более интересный параметр — SQL Mail. Если на компьютере установлена система электронной почты, то здесь можно задать профиль почты, тогда ваш экземпляр SQL Server 2000 может быть конфигурирован для отправки и получения электронных сообщений. Сервер можно сконфигурировать таким образом, что при возникновении ошибки он отправит электронное письмо с уведомлением об этом. Прекрасно, не правда ли? Эта возможность рассматривается несколько позже.
- Во вкладке Memory задается объем памяти, который может использовать SQL Server. По умолчанию SQL Server 2000 использует всю имеющуюся память. В конце концов, раз она есть, то почему бы ее не использовать? Однако может оказаться, что другие приложения выполняются слишком медленно. В этом случае можно настроить SQL Server 2000 на использование только указанного объема памяти.



Ограничение объема памяти, используемого SQL Server 2000, может повлиять на выполнение приложений, поэтому хорошо подумайте, прежде чем изменять эту опцию. Сервер использует всю имеющуюся память, но он не жадный. Если другие приложения требуют память для выполнения своих задач, SQL Server 2000 освободит ее для них. Весьма любезно, не правда ли?

На заметку

Сервер использует всю имеющуюся память. Поэтому если на компьютере установлено несколько экземпляров SQL Server 2000, то они будут размещать и освобождать ресурсы памяти по мере необходимости. Каждый экземпляр SQL Server 2000 считает, что другой экземпляр — обыкновенное приложение, поэтому не следует устанавливать на компьютере слишком много экземпляров SQL Server 2000.

- Параметры вкладки Processor также могут потребовать конфигурирования (в зависимости от операционной системы). В этой книге рассматривается работа сервера только под управлением Windows 98. Однако весьма вероятно, что разрабатываемой системе придется работать под управлением Windows NT/2000 в многопроцессорной среде. В таком случае вернитесь к этому разделу и установите опции, учитывающие специфику мультипроцессорности Windows NT/2000.

На этом закончим рассмотрение параметров конфигурирования сервера. Как отмечалось ранее, таких параметров довольно много, но вам предстоит ознакомиться с ними самостоятельно, руководствуясь материалом главы 15, "Самостоятельное исследование SQL Server 2000".

Что теперь? Рассмотрим некоторые более общие задачи, которые должен выполнять администратор базы данных.

Выполнение общих задач администрирования

Помимо проверки параметров сервера и резервного копирования, администратор базы данных выполняет множество других задач. Если бы работа администратора была слишком легкой, то вы не захотели бы этим заниматься! Насколько утомительна эта работа?

Как уже отмечалось, вы должны обеспечить хорошую работу сервера, а также быстрое и эффективное выполнение приложений баз данных. Есть хорошая новость: большинство задач поддержки баз данных могут выполняться автоматически. Про-

думав состав задач поддержки и время их выполнения, можно создать набор задач с расписанием, и тогда они будут автоматически выполняться в указанное время.

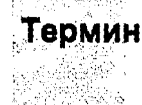
Рассмотрим создание задачи с расписанием, которая будет выполнять резервное копирование базы данных ночью, но вам при этом не придется вставать в 3 часа, чтобы выполнить эту работу. Вам бы это понравилось? Возможно, вашему новому боссу и понравилось бы, но вам — вряд ли.

Создание расписаний задач

Что такое задача с расписанием? Это похоже на напоминания в Microsoft Outlook. Можно установить автоматическое выполнение задачи в назначенное время, например задать резервное копирование базы данных в 3 часа ночи. Сделать это довольно просто, и параметры конфигурирования расписания каждой задачи весьма разнообразны.

На заметку Подобно большинству средств SQL Server 2000, задачи с расписанием можно создать с помощью как *Enterprise Manager*, так и языка Transact-SQL. Сейчас воспользуемся графическим интерфейсом пользователя *Enterprise Manager*, но, если вы захотите узнать, как это сделать с помощью команд Transact-SQL, обратитесь к системе справки SQL Server 2000 (поищите раздел *sp_add_jobschelule*).

1. Запустите Enterprise Manager, откройте папку Management, а затем папку SQL Server Agent. В ней вы увидите три опции.
   - Alerts (Извещения). Здесь можно задать определенные действия при появлении указанной ошибки.
   - Operators (Операторы электронной почты). Это учетные записи пользователей, которым при возникновении аварийной ситуации можно отправить электронное письмо, сетевое оповещение (для Windows NT/2000) или даже сообщение на пейджер.
   - Jobs (Задачи). Здесь находятся задачи, содержащие один или больше шагов выполнения. Эти шаги являются операторами Transact-SQL, которые SQL Server может выполнить.
2. Выделите папку Jobs, щелкните правой кнопкой мыши и из появившегося контекстного меню выберите команду New Job (рис. 11.2).
3. Мы создаем задачу резервного копирования базы данных SQLSpyNet, поэтому присвойте ей соответствующее имя, например *Database Backup for SQLSpyNet*. Задачам рекомендуется давать описательные имена, чтобы каждый (и вы в том числе), глядя на имя, мог понять, что делает эта задача. Максимальная длина имени — 128 символов, этого вполне достаточно.
4. Далее необходимо подключить задачу. Для этого установите флажок Enabled (Активизированная). Параметр Target Local Server (Целевой локальный сервер) задает выполнение задачи текущим экземпляром SQL Server 2000. Если есть *связанные серверы* (т.е. соединенные в сеть), можно создать задачу, выполняющуюся на этих связанных серверах, несмотря на то что она активизирована только на нашем сервере.



Связанные серверы — это источники данных, с которыми может общаться SQL Server 2000 (например, другие серверы SQL Server, Excel или Access). На эти серверы можно посылать запросы, возвращающие данные или выполняющие там определенные действия.

Рис. 11.2. Выбор команды New Job для создания новой задачи в SQL Server 2000

При выборе команды New Jobs появится окно, показанное на рис. 11.3.

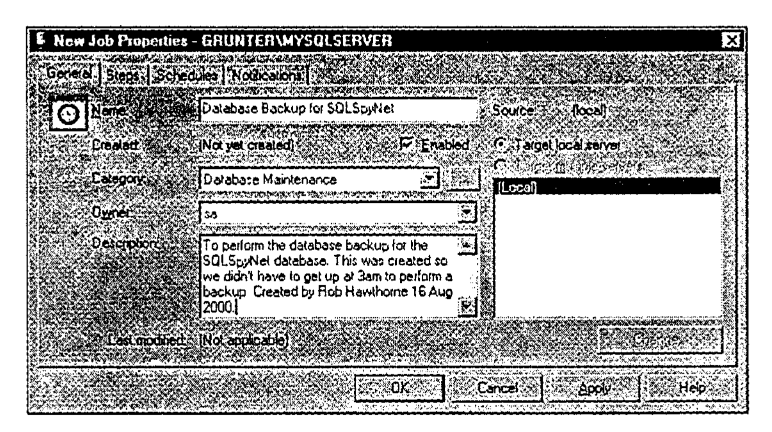


Рис. 11.3. Установка общих свойств новой задачи

5. Чтобы лучше организовать задачи, им можно присвоить определенные категории (опция Category). Присвойте задаче категорию Database Maintenance (Поддержка базы данных). Если задач много, то их можно отсортировать по категориям, поэтому присвоение категории всегда полезно.

На заметку Если щелкнуть на кнопке с троеточием (...), то можно увидеть все задачи этой категории.

6. Далее задаче можно присвоить владельца. Если задачей владеет другой пользователь, нужно убедиться, что у него есть все права доступа, необходимые для выполнения этой задачи. Например, может оказаться, что вы присвоили задаче владельца `SQLSpyNetUser`, однако он не может ее выполнить, потому что не имеет прав создания резервной копии базы данных. Сделайте владельцем задачи пользователя с учетной записью `sa`.
7. Добавим для нашей задачи ее описание. Оно позволит определить, каковы функции задачи, когда и кем она была создана. В будущем, когда вы забудете об этой задаче, описание поможет определить, нужна ли она еще. Максимальная длина описания — 512 символов. Введите примерно такой текст в поле ввода Description (Описание): **Задача выполняет резервное копирование базы данных. Она была создана, чтобы не вставать для этого в 3 часа ночи. Создал задачу *имя*. Дата создания — *дата*.**
8. Далее, создадим шаги задачи, т.е. определим, что, собственно, она должна сделать. Для этого активизируйте вкладку Steps (Шаги), как показано на рис. 11.4.
9. Щелкните на кнопке New. Это позволит создать первый (в данном примере единственный) шаг, который SQL Server 2000 выполнит для резервного копирования базы данных. После щелчка на кнопке появляется окно, показанное на рис. 11.5.

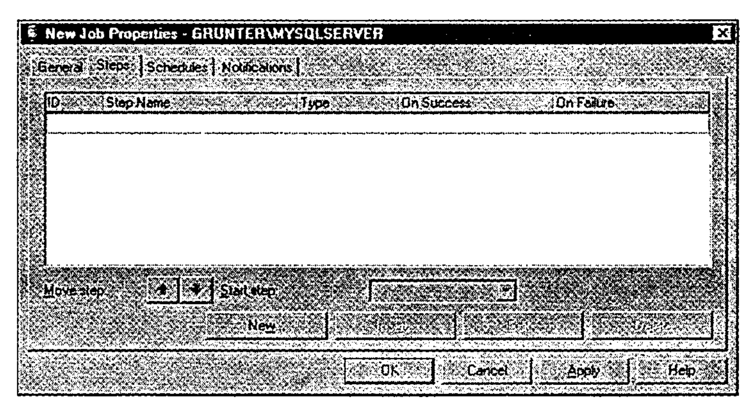


Рис. 11.4. Установка шагов новой задачи, выполняющей резервное копирование базы данных

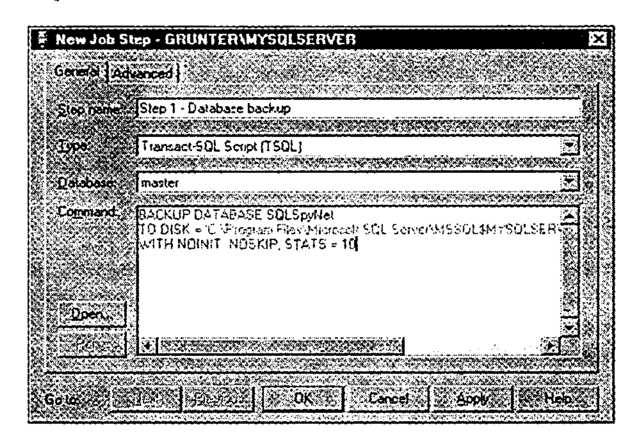


Рис. 11.5. Задание команд SQL Server 2000, которые должна выполнить новая задача

10. Сначала присвойте шагу имя, например **Step 1 Database Backup**. Имя шага должно быть уникальным, его длина не должна превышать 128 символов.

11. Далее задайте тип команд, которые SQL Server 2000 выполнит на этом шаге. В большинстве случаев это будут команды Transact-SQL, однако можно задать выполнение сервером команд операционной системы, например пакетных файлов, или каких-либо сценариев. Сейчас оставьте активной опцию Transact-SQL.

12. Затем укажите базу данных, в которой нужно выполнить задачу.

13. И наконец, введите команду, которую должен выполнять SQL Server 2000. Используйте оператор копирования, описанный в главе 10, "Обеспечение доступности данных". Вот как хорошо знать и операторы Transact-SQL, и графический интерфейс пользователя! Иначе пришлось бы не поспать несколько ночей.

```
1:   BACKUP DATABASE SQLSpyNet
1a:  TO DISK = 'C:\Program Files\Microsoft SQL Server\
     ↳MSSQL$MYSQLSERVER\BACKUP\SQLSpyNetJob.bak'
1b:  WITH NOINIT, NOSKIP, STATS = 10
```

Единственное изменение — это имя создаваемого файла. На этот раз имя файла имеет суффикс Job.

На заметку Суффиксы имен файлов не обязательны. Здесь суффикс нужен только для того, чтобы напоминать, что содержит файл резервной копии. Это довольно удобно.

Кнопка Open позволяет загрузить в окно ввода ранее созданный сценарий Transact-SQL. С помощью кнопки Parse можно проверить правильность синтаксиса введенной команды. Щелкните на ней сейчас, чтобы потом не жалеть.

14. Во вкладке Advanced (Дополнительно) можно задать, какое действие должен выполнить SQL Server 2000 в случае успеха или неудачи выполнения этого шага. Значение опций этого окна по умолчанию нас вполне устраивает, поэтому оставим все как есть:

- действие в случае успеха — переход к следующему шагу;
- попытки повтора — 0;
- действие в случае неудачи — выход из задачи с соответствующим сообщением.

  Щелкните на кнопке OK. Вы увидите, что задача появилась в поле со списком Steps (Шаги). Если бы процесс имел насколько шагов, то здесь можно было бы переставить их местами, изменив таким образом последовательность выполнения.

15. Теперь нужно указать время выполнения задачи. Активизируйте вкладку Schedules. Щелкните на кнопке New Schedule. Появившийся экран будет выглядеть примерно так, как показано на рис. 11.6.

16. Как и другие имена, имя расписания должно быть описательным и уникальным, а его длина не должна превышать 128 символов. Введите что-нибудь вроде **3 часа ночи. Расписание резервного копирования базы данных SQLSpyNet**. Обязательно установите флажок Enabled.

17. Теперь нужно задать расписание. Кратко перечислим его опции.

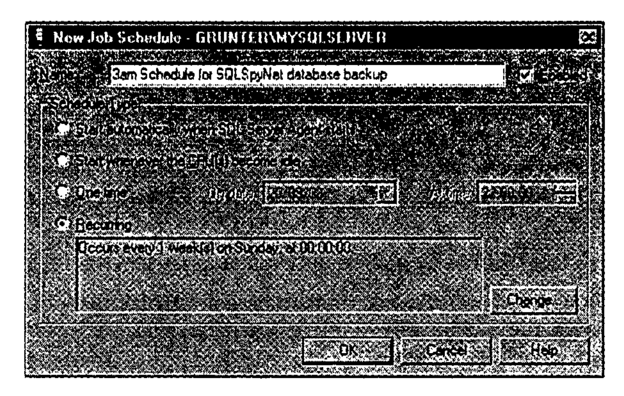


Рис. 11.6. Составление расписания выполнения новой задачи

- Start automatically when SQL Server Agent starts (Автоматическое выполнение при запуске SQL Server Agent). Задает автоматическое выполнение задачи при каждом запуске утилиты SQL Server Agent. Подробнее вопрос запуска рассматривается в главе 10, "Обеспечение доступности данных".
- Start whenever CPU becomes idle (Автоматическое выполнение при незанятом процессоре). Для экземпляра SQL Server 2000 можно задать условие незанятости процессора. Тогда конфигурируемая задача будет автоматически запускаться при незанятом процессоре.
- One time (Один раз). Если включена эта опция, то задача будет выполнена один раз в указанное время.
- Recurring (Повторение). Позволяет установить многократное выполнение задачи в указанные моменты времени. Эту опцию мы сейчас будем настраивать.

Установите переключатель Recurring и щелкните на кнопке Change. Появится окно, показанное на рис. 11.7.

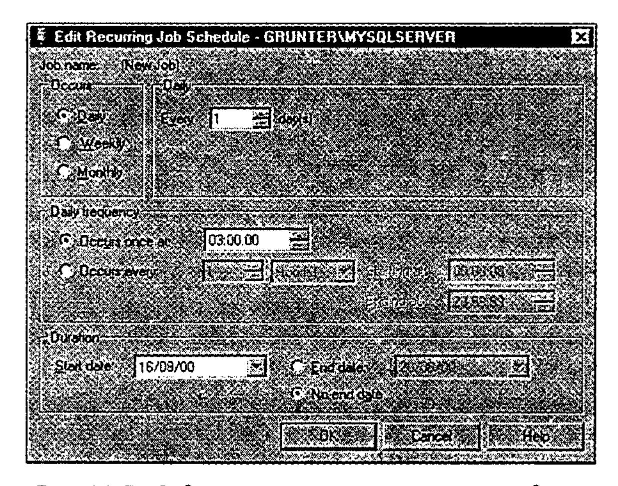


Рис. 11.7. Задание времени выполнения задачи в SQL Server 2000

18. Опции этого диалогового окна очевидны, особенно если вам приходилось устанавливать опции повторения в Outlook. Сейчас установите в нем следующие параметры:

- Occurs — Daily (Выполнять — задаются дни);
- Daily — Every 1 day (Каждый день);
- Daily frequency (Как часто в течение дня) — 3:00:00 a.m.;
- Duration (Продолжительность): Start date (Дата начала) — установите здесь текущую дату; End date — не указывайте.

19. Щелкните на кнопке OK. Вы вернулись в окно New Job Schedule. Еще раз щелкните на кнопке OK. В окне задачи вы видите созданное только что расписание.

Так просто? Но что делать, если при автоматическом выполнении будут происходить ошибки? Рад, что вы стараетесь все предусмотреть. Никогда нельзя гарантировать, что все будет работать идеально, поэтому следует задавать действия системы в случае ошибки. Для этого предназначена вкладка Notifications (Уведомления). Активизируйте ее (рис. 11.8).

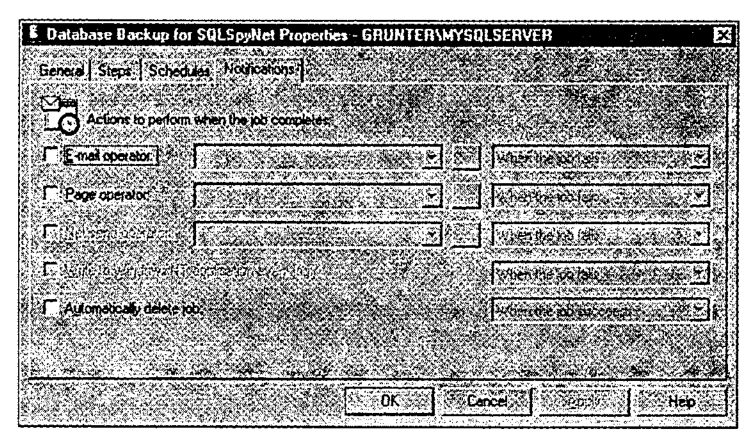


Рис. 11.8. Задание действий системы в случае успешного или неудачного выполнения задачи

В этой вкладке наиболее часто используемая опция — запись в журнал событий Windows NT. Однако, к сожалению, мы используем операционную систему Windows 98, поэтому не можем внести запись в журнал событий NT (его здесь нет). Так что же делать?

Введем понятие оператора. С помощью оператора для SQL Server 2000 задается конкретный получатель, которому можно отправить сообщение электронной почты или сообщение на пейджер (с помощью дополнительного программного обеспечения пейджинговой связи). Но это рассматривается несколько позже.

Если сейчас щелкнуть на кнопке OK, то новая задача будет сохранена и готова к выполнению в 3 часа ночи. Однако здесь нужно обратить внимание на ряд обстоятельств.

- Для выполнения задачи компьютер должен быть включен. Это кажется очевидным, однако мне приходилось слышать много историй о том, как задачи были конфигурированы, но не выполнялись из-за выключенного компьютера. Поэтому не лишним будет еще раз напомнить об этом.

- Регулярно проверяйте, выполнялась ли задача. В папке Jobs можно увидеть список задач, время их последнего выполнения и результат выполнения (успех или неудача). Необходимо также проверять, сохранен ли файл с резервной копией. Если он существует и дата его создания правильная, значит, резервное копирование успешно выполнено.
- Создайте вторичную базу данных и иногда восстанавливайте в ней резервную копию. Таким образом вы сможете убедиться, что резервное копирование выполняется правильно.

Итак, господа, мы закончили еще один раздел об администрировании. Теперь рассмотрим операторы.

Вызов операторов SQL Server

С помощью оператора можно определить человека, который будет уведомлен при наступлении определенных событий. Например, если резервное копирование закончилось неудачно, то неплохо было бы немедленно получить электронное письмо, уведомляющее об этом.

Чтобы использовать операторы, необходимо выполнить некоторые требования.

- Экземпляр SQL Server 2000 должен обладать способностью отправлять электронные письма. Это свойство устанавливается рассмотренными ранее опциями конфигурирования сервера или в диалоговом окне свойств утилиты SQL Server Agent. Щелкните правой кнопкой на пиктограмме SQL Server Agent и выберите команду Properties (Свойства). Более подробная информация об установке профилей почты содержится в документации Microsoft Exchange или Outlook.
- Возможность отправки сетевых сообщений поддерживается только при использовании Windows NT/2000.
- Опция Pager e-mail name содержит электронный адрес, по которому отправляется уведомление на пейджер, поэтому для пейджера также должен быть установлен профиль. Для преобразования электронного письма в пейджинговое нужно дополнительное программное обеспечение.

На заметку Между персональным выпуском SQL Server 2000 под управлением Windows 98 и SQL Server под управлением Windows NT/2000 разница невелика. Однако в папке *SQL Server Agent* она становится очевидной. Под управлением Windows NT/2000 можно конфигурировать некоторые дополнительные опции. Учитывайте это при переходе на другую систему или при настройке системы для многопользовательской среды.

Поскольку мы работаем в операционной системе Windows 98, то не можем устанавливать опцию Net send address. Рассмотрим, как настраивать отправку уведомления.

1. Выделите папку Operators (в папке SQL Server Agent). Щелкните правой кнопкой мыши и выберите команду New Operator (рис. 11.9).
2. Сначала задайте имя оператора. Оно должно быть уникальным, т.е. никакой другой оператор не может иметь это же имя. Длина имени не должна превышать 128 символов.
3. Рекомендуется присваивать оператору легко узнаваемое имя. Здесь я дал ему мое собственное имя Rob.

4. Теперь установите параметр E-mail name. Это имя пользователя, которому отправляется письмо. Также можно использовать электронный адрес в квадратных скобках, например [SMTP:someone@somewhere.com]. Щелкнув на кнопке Test, можно отправить письмо оператору (скорее, имитировать отправку).

Рис. 11.9. Выбор команды New Operator

Появится диалоговое окно, показанное на рис. 11.10.



Если тестовая отправка письма завершается неудачей, проверьте профиль электронной почты, с которым был конфигурирован сервер. Если имя неправильное, то сервер не сможет отправить вам сообщение.

5. Конфигурируйте параметр Pager e-mail name. В это поле можно ввести электронный адрес оператора, которому вы хотите отправлять пейджинговые сообщения. Как и в случае параметра E-mail name, можно протестировать отправку сообщения, щелкнув на кнопке Test. Если используется параметр Pager e-mail name, то в группе Pager on duty schedule можно задать расписание отправки сообщений.

6. Присвоив оператору электронный адрес, активизируйте вкладку Notifications. В ней можно задать события, при наступлении которых отправляется уведомление оператору.

   Сначала на экране присутствуют две опции: Alerts (Оповещения) и Jobs (Задачи).

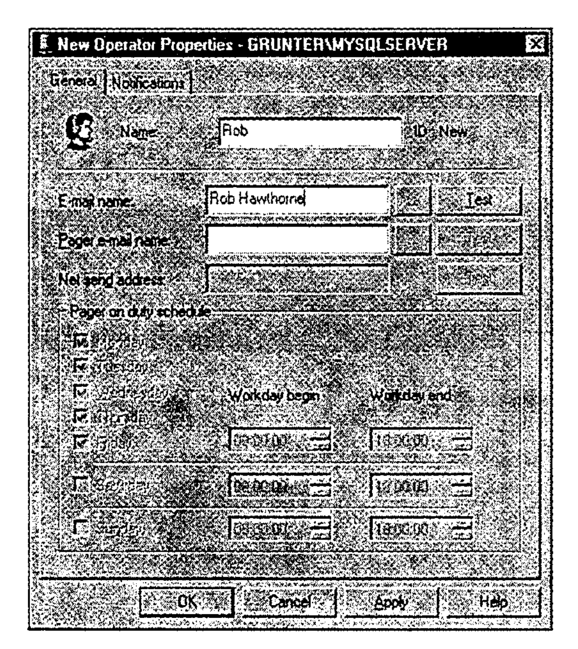


Рис. 11.10. Диалоговое окно New Operator Properties

- Опция Alerts предназначена для сообщений об ошибках, сгенерированных системой, например DEMO: Full tempdb. Более подробно оповещения рассматриваются далее в главе.
- Опция Jobs позволяет просмотреть задачи, которые будут уведомлять оператора о выполнении. Сейчас у нас еще нет присвоенных операторам уведомлений.

1. Теперь у нас есть оператор и задача. Для отправки уведомления нужно назначить этот оператор задаче. Щелкнув на кнопке OK, закройте диалоговое окно создания нового оператора.
2. Откройте созданную задачу. Для этого откройте папку Jobs и дважды щелкните на задаче Database Backup for SQLSpyNet. Активизируйте вкладку Notifications (рис. 11.11).
3. Во вкладке Notifications можно задать действие, выполняемое в случае успешного или неудачного завершения задачи. Мы хотим получать уведомление в случае ее неудачного завершения. Установите флажок E-mail operator и из раскрывающегося списка выберите нужный оператор (в примере книги это Rob). Далее необходимо указать событие, при котором отправляется уведомление. Из раскрывающегося списка выберите When the job fails (При неуспехе). Щелкните на кнопке OK.

Дело сделано! Теперь при неуспехе задачи указанный оператор будет уведомлен сообщением электронной почты. Можете себе представить, насколько огромны возможности этих средств в реальных системах бизнеса.

Однако есть еще одно средство, с помощью которого систему администрирования сервера можно сделать более гибкой: это оповещения.

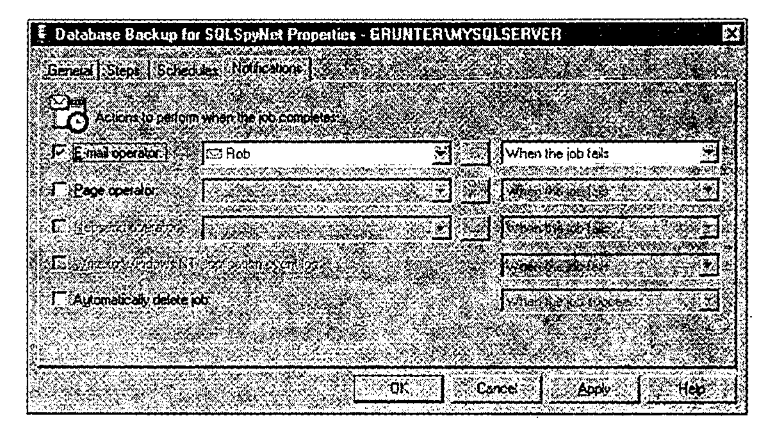


Рис. 11.11. Назначение задачи оператору с целью уведомления

Использование оповещений

Оповещение порождает событие, когда в SQL Server 2000 некоторое заданное условие становится истинным. Предположим, что некоторая "тяжелая" транзакция неожиданно существенно увеличивает размер базы данных. Мы можем сконфигурировать SQL Server 2000 таким образом, что при возникновении подобной ситуации сервер отправит нам электронное письмо и выполнит задачу, создающую резервную копию журнала транзакций. Потрясающая гибкость!

Поскольку наша база данных все еще довольно простая, мы не будем создавать оповещение. Однако вы должны помнить, что при работе с реальной базой данных оповещения помогут сделать систему гораздо более гибкой. Подробнее оповещения описаны в справочной системе, работа с которой рассматривается в главе 15, "Самостоятельное исследование SQL Server 2000".

Журналы SQL Server

В главе 10, "Обеспечение доступности данных", описано несколько типов журналов SQL Server, используемых для выполнения различных задач. В этом разделе рассматривается важнейшее средство поддержки SQL Server 2000 — журналы активности SQL Server.

Эти журналы отслеживают и контролируют все, что происходит в данном экземпляре SQL Server 2000. При каждой остановке и запуске SQL Server 2000 сервер запускает новый журнал активности. Следовательно, при каждой загрузке системы создается новый журнал.

В окне Enterprise Manager журналы расположены в папке Management\SQL Server Logs (рис. 11.12).

Дважды щелкните на одном из журналов, и вы увидите события, записанные сервером.

Для тех, кто работал с Windows NT/2000, вид журналов покажется очень знакомым. Подробную информацию о каждом событии можно получить, дважды щелкнув на нем в журнале (рис. 11.13).

Просматривая журнал, вы увидите много событий, возникающих при запусках базы данных SQLSpyNet. Это объясняется тем, что при создании базы данных установлена опция Auto Close.

SQL Server Enterprise Manager - [Console Root\Microsoft SQL Servers\SQL Server Group\GRUNTER\MYSQLSERVER (Win...

Console Window Help

Action View Tools

Tree

Console Root
Microsoft SQL Servers
SQL Server Group
GRUNTER\MYSQLSERVER
Databases
Data Transformation Ser
Management
SQL Server Agent
Alerts
Operators
Jobs
Backup
Current Activity
Database Maintenan
SQL Server Logs
Current - 10/15/
Archive #1 - 09/
Replication
Security
Support Services
Meta Data Services

Current - 10/15/2000 21:15 (2597 Byte) 29 Items

Date	Source	Message
2000-10-15 21:01:35.01	server	Microsoft SQL Server 2000 - 8.00.194 (Intel X86) ...
2000-10-15 21:01:35.23	server	Copyright (C) 1988-2000 Microsoft Corporation.
2000-10-15 21:01:35.26	server	All rights reserved.
2000-10-15 21:01:35.28	server	Server Process ID is -176185.
2000-10-15 21:01:35.31	server	Logging SQL Server messages in file 'd:\mssql2000\MSSQL$MYS(
2000-10-15 21:01:35.45	server	SQL Server is starting at priority class 'normal'(1 CPU detected)
2000-10-15 21:01:36.60	server	SQL Server configured for thread mode processing.
2000-10-15 21:01:36.73	server	Using dynamic lock allocation. [500] Lock Blocks, [1000] Lock Ow
2000-10-15 21:01:37.38	spid3	Starting up database 'master'.
2000-10-15 21:01:39.45	spid5	Starting up database 'model'.
2000-10-15 21:01:41.93	server	SQL server listening on TCP, TCP, Shared Memory.
2000-10-15 21:01:42.01	server	SQL server listening on 127.0.0.1:1026, 127.0.0.1:1027.
2000-10-15 21:01:42.13	server	SQL Server is ready for client connections
2000-10-15 21:01:42.48	spid3	Server name is 'GRUNTER\MYSQLSERVER'.
2000-10-15 21:01:42.70	spid3	Skipping startup of clean database id 4
2000-10-15 21:01:42.75	spid3	Skipping startup of clean database id 5
2000-10-15 21:01:42.82	spid3	Skipping startup of clean database id 6
2000-10-15 21:01:43.46	spid5	Clearing tempdb database.
2000-10-15 21:01:47.54	spid5	Starting up database 'tempdb'.
2000-10-15 21:01:48.85	spid3	Recovery complete.
2000-10-15 21:06:37.39	spid51	Using 'xpstar.dll' version '2000.80.194' to execute extended stored (
2000-10-15 21:06:39.70	spid51	Starting up database 'msdb'.
2000-10-15 21:07:06.44	spid51	Configuration option 'show advanced options' changed from 0 to 1.
2000-10-15 21:07:06.44	spid51	Error: 15457, Severity: 0, State: 1
2000-10-15 21:07:07.33	spid51	Using 'xplog70.dll' version '2000.80.194' to execute extended store

Рис. 11.12. Журналы активности в Enterprise Manager

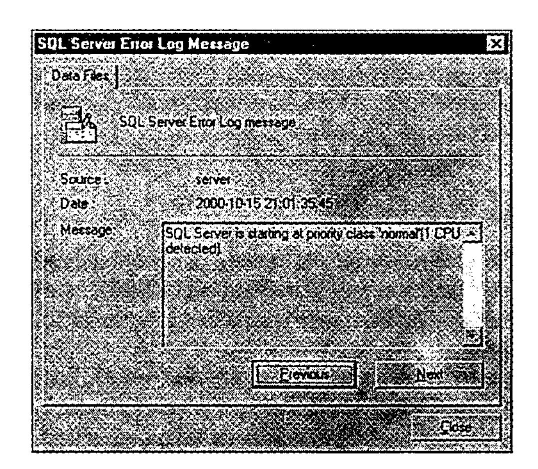


Рис. 11.13. Просмотр информации о событиях в журналах активности

Как видно из журналов, открывать базу данных при каждом запросе для сервера несколько обременительно. Однако для настольной базы данных эта опция достаточно эффективна и полезна. При переходе к многопользовательской системе рекомендуется эту опцию не применять.

Когда в SQL Server 2000 происходит ошибка, она протоколируется в файле журнала. Вы часто будете получать от SQL Server 2000 сообщения об ошибках с указанием обратиться к журналам для получения более подробной информации. Получив сообщение об ошибке, вы можете также обратиться к справочной системе (или к дру-

гим многочисленным источникам, описанным в главе 15, "Самостоятельное исследование SQL Server 2000").

Однако, скорее всего, журналы вам не понадобятся, пока в SQL Server 2000 не произойдет ошибка. Поэтому вернемся к вопросам поддержки.

Проверка целостности данных

Какова следующая задача администрирования? Мы должны периодически проверять целостность нашей базы данных. Эти проверки помогут обеспечить целостность данных. Чем раньше будут обнаружены проблемы, тем легче будет их решить. Можно проводить логические и физические проверки базы данных, таблиц, индексов и групп файлов.

Операторы проверки целостности группируются в четыре категории.

- Операторы поддержки. Используются для проверки целостности базы данных, индексов и таблиц.
- Различные операторы. Позволяют, например, конфигурировать блокировку на уровне строк или удалять из памяти оставшиеся ненужные динамические библиотеки `.dll`.
- Операторы достоверности. Подтверждают непротиворечивость и удовлетворительное состояние объектов базы данных (таблиц, представлений и т. д.).
- Операторы состояния. Информируют о состоянии объектов базы данных, например насколько фрагментирован индекс.

Главный оператор проверки целостности — `DBCC CHECKDB`. Это оператор достоверности. Вам придется довольно часто выполнять его. Оператор возвращает отчет о текущем состоянии базы данных, т.е. число страниц на диске, используемых таблицами.

Существует большое количество операторов `DBCC`, включая проверки дефрагментации индексов, восстановления базы данных и др. Эти операторы предоставляют много разных возможностей. Подробная информация об этом содержится в справочной системе.

В целом можно с уверенностью утверждать, что SQL Server 2000 достаточно надежно поддерживает правильную работу баз данных. Периодически выполняемые проверки нужны в большей степени для нас самих: чтобы быть спокойными.

Администратор базы данных должен выполнять регулярную проверку с помощью `DBCC` для того, чтобы быть уверенным, что все работает правильно. Компания Microsoft проделала в этой области огромную работу для того, чтобы единственной причиной периодических проверок было желание удостовериться в правильной работе сервера, а не реальная необходимость что-либо исправлять.

Выполним оператор `DBCC CHECKDB`. Поскольку это оператор Transact-SQL, то, как вы уже догадались, необходимо запустить Query Analyzer.

В окне Query Analyzer введите код листинга 11.1.

Листинг 11.1. Выполнение оператора DBCC для проверки целостности базы данных

Код для запуска

```
1: DBCC CHECKDB('SQLSpyNet')
```

Запрос возвратит большое количество информации о базе данных SQLSpyNet, включая количество строк на каждой странице каждой таблицы базы данных.

Главное, что нужно сделать при этой проверке, — быстро просмотреть результат и отметить, есть ли ошибки. В листинге результата они хорошо видны, однако скорее всего их не будет.

Если ошибки все же есть, то найдите в справочной системе сведения, указывающие, как устранить возникшие проблемы.

На заметку

Процедура DBCC CHECKDB — одна из лучших, потому что проверяет целостность базы данных и обнаруживает подавляющее большинство возможных ошибок. Она выполняет задачи процедур DBCC CHECKALLOC и DBCC CHECKTABLE, поэтому нет необходимости выполнять их.

В предыдущих версиях (SQL Server 6.5 и более ранних) процедура DBCC CHECKDB выполнялась несколько часов. Компания Microsoft приложила немало усилий, чтобы сократить это время, не снижая эффективности. Однако процедура все еще довольно затратная. Поэтому не выполняйте DBCC CHECKDB в периоды наибольшей загрузки сервера, особенно на больших базах данных.

Мы рассмотрели, как создавать резервные копии баз данных и как установить автоматическое выполнение этой операции в нерабочее время. Кроме того, описано выполнение проверок целостности базы данных, которые также должны происходить в "тихое" время. Но если выполнить проверку автоматически в нерабочее время, то как можно просмотреть ее результат? Как совместить эти задачи? Все это можно сделать с помощью мастера плана поддержки (Maintenance Plan Wizard).

Создание плана поддержки целостности и доступности базы данных

Термин

План поддержки позволяет объединить проверки целостности и резервное копирование базы данных в одно целое. В план поддержки можно также включить обновление статистики базы данных.

Если статистика базы данных не устарела, то эффективность работы SQL Server 2000 будет выше, потому что сервер сможет правильно решить, например, использовать ли при выполнении запроса индексы или применить простое сканирование таблицы. Сервер сам периодически обновляет статистику. Но с момента последнего обновления могло пройти некоторое время, поэтому принудительное обновление статистики может повысить эффективность работы SQL Server 2000.

Мастер Maintenance Plan Wizard способен уменьшить размер файлов данных путем удаления неиспользуемых страниц данных. Это значительно сэкономит дисковое пространство.

Мастер позволяет также конфигурировать некоторые другие параметры. Но перейдем от слов к делу!

Создание плана поддержки с помощью мастера *Maintenance Plan Wizard*

Для запуска мастера Maintenance Plan Wizard выполните ряд действий.

1. Щелкните правой кнопкой на базе данных SQLSpyNet и выберите команду All Tasks⇨Maintenance Plan (рис. 11.14). Произойдет запуск мастера, как показано на рис. 11.15.

Рис. 11.14. Запуск мастера Maintenance Plan Wizard из окна Enterprise Manager

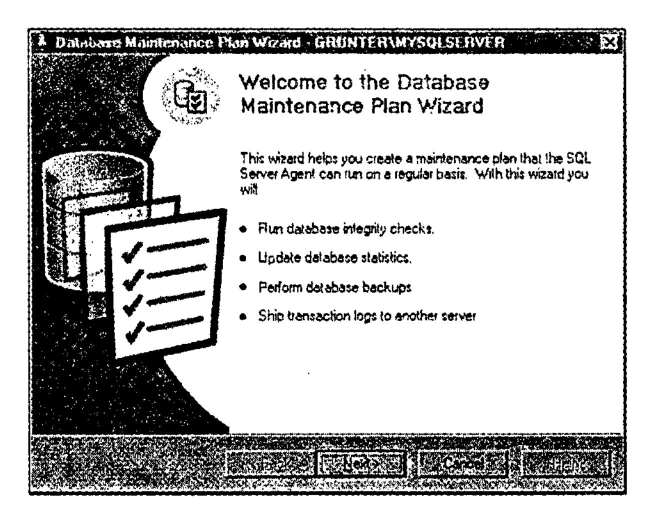
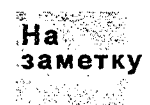


Рис. 11.15. Начальное окно мастера Maintenance Plan Wizard

2. В этом окне опций нет, просто щелкните на кнопке Next. Появится окно выбора баз данных, для которых мы хотим создать план поддержки.



До сих пор выполнялась поддержка только *SQLSpyNet*, однако системные базы данных тоже нуждаются в поддержке. Если они работают неэффективно, то наша база данных тоже не сможет работать эффективно. Системные базы данных — ключевой пункт всех операций с данными.

3. В этом окне мастера выделите не только базу данных SQLSpyNet, но и базы данных model, master и msdb (рис. 11.16).

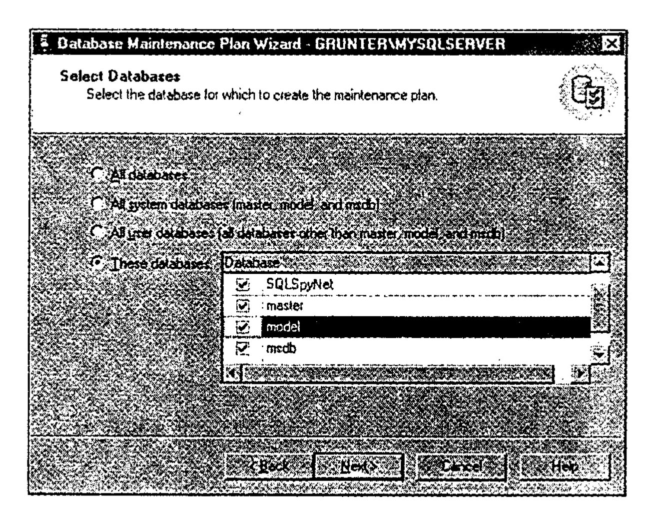


Рис. 11.16. Выбор баз данных, для которых нужно выполнять задачи поддержки

Другие базы данных (Northwind и Pubs) — это примеры баз данных, поставляемых с SQL Server 2000. Возможно, они у вас даже не установлены. Щелкните на кнопке Next.

4. Появляется окно, показанное на рис. 11.17.

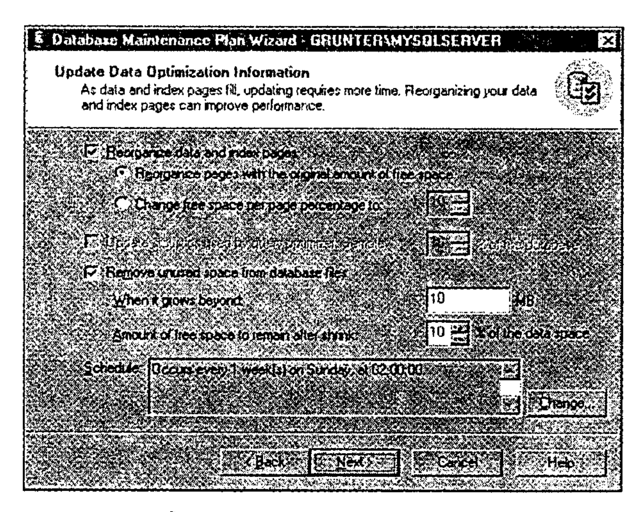


Рис. 11.17. Здесь можно задать обновление индексов и страниц данных

5. В этом окне можно конфигурировать обновление и поддержку индексов. Здесь же можно указать, что нужно сделать со свободным пространством в страницах данных. Перечислим представленные в этом окне опции.

   - Reorganize data and index pages. Если этот флажок установлен, то SQL Server 2000 удалит и создаст заново все индексы таблиц, используя новый или исходный фактор заполнения. Выберите исходный фактор заполнения, установив переключатель Reorganize pages with the original amount of free space (Реорганизовать таблицы с исходным количеством свободного пространства).
   - Update statistics used by query optimizer sample (Обновить статистику, используя образец для программы оптимизации запросов). Задает часть данных (в процентах), которая будет использована в качестве образца для вычисления статистики. Программа оптимизации использует эту информацию для выработки оптимального плана выполнения запроса.
   - Remove unused space from database files (Удалить неиспользуемое пространство из файлов базы данных). Если этот флажок установлен, то сервер таким образом уменьшает размер файлов данных. В поле When it grows beyond (Когда он станет больше чем) можно задать минимальный размер файла, подлежащего сокращению. Мы работаем с базой данных на небольшом компьютере, а потому установите эту опцию в 10 Мбайт. Оставьте значение параметра Leave the amount of free space to remain after shrink (Оставить количество свободного пространства после сокращения) по умолчанию — 10%.
   - Schedule (Расписание). Эта опция позволяет установить дату и время выполнения процедур оптимизации. Если щелкнуть на кнопке Change, то можно будет изменить расписание. Появляющееся при этом диалоговое окно установки расписания аналогично окну установки расписания задач, рассмотренному в предыдущем разделе. Сейчас в примере книги установлено следующее расписание (см. рис. 11.17): процедуры оптимизации выполняются еженедельно по воскресеньям в два часа ночи.

6. Щелкните на кнопке Next. Появится диалоговое окно с опциями проверки целостности базы данных (рис. 11.18).

7. Если установить флажок Check database integrity (Проверять целостность базы данных), то становятся доступными несколько опций, позволяющих конфигурировать данную программу поддержки.

   - Include indexes (Включая индексы). Если активизирован этот параметр, то процедуре проверки подвергаются также индексы. Установите этот переключатель.
   - Attempt to repair any minor problems (Попытаться решать небольшие проблемы). Если активизирован этот параметр, то SQL Server 2000 попытается самостоятельно исправить небольшие нарушения целостности. Microsoft рекомендует устанавливать этот флажок.
   - Perform these tests before doing backups (Выполнять эти проверки перед резервными копированиями). Если этот параметр активизирован, указанные проверки будут выполняться перед каждым резервным копированием. Если проверка обнаружит нарушение целостности, то резервное копирование выполняться не будет. Установите этот флажок.
   - Schedule. Этот параметр позволяет установить дату и время выполнения процедур проверки целостности. Установлено выполнение процедур по воскресеньям в час ночи.

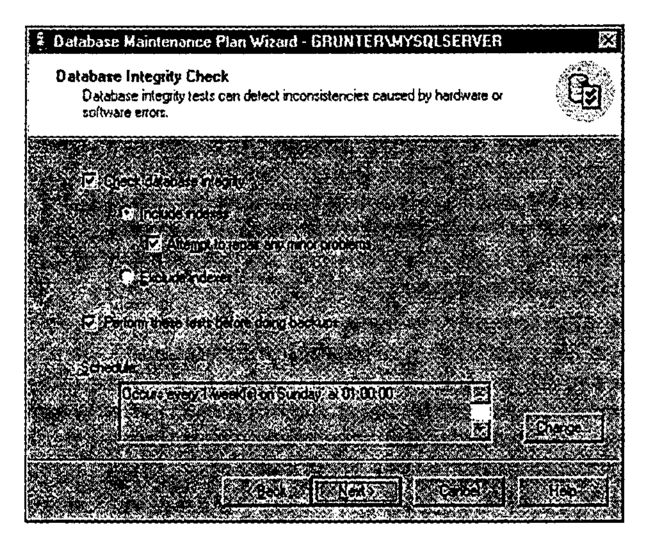


Рис. 11.18. Установка опций проверки целостности баз данных и индексов

Установив необходимые параметры, щелкните на кнопке Next. Появится диалоговое окно создания плана резервного копирования (рис. 11.19).

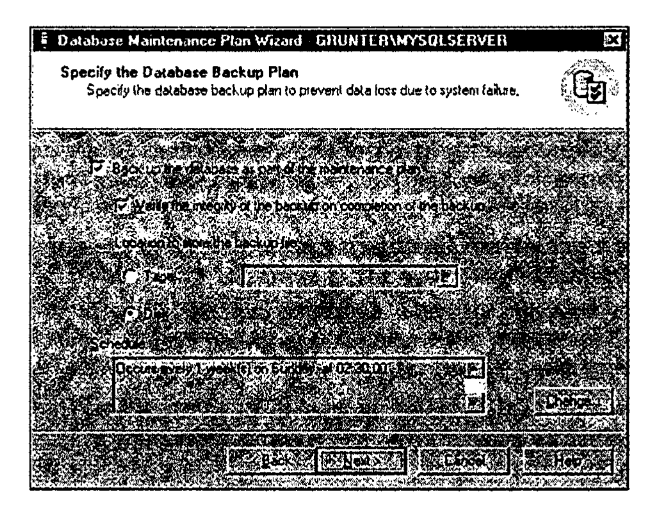


Рис. 11.19. Диалоговое окно создания плана резервного копирования



Как вы помните, мы уже создали задачу, выполняющую резервное копирование каждый день в три часа ночи. Сейчас создадим план резервных копирований, выполняющихся по воскресеньям в 2.30. В реальной жизни оба процесса могут оказаться уместными: частичное копирование каждый день и полное копирование каждую неделю.

8. Установите флажок Back up the database as part of the maintenance plan (Резервное копирование базы данных как часть плана поддержки). Это сделает доступными несколько опций.

- Verify the integrity of the backup on completion of the backup (Проверять целостность резервной копии при завершении копирования).
- Location to store the backup file (Место сохранения резервной копии). Здесь необходимо указать магнитную ленту или диск. Накопителя на магнитной ленте у вас, конечно же, нет, поэтому укажите диск.
- Schedule. Установите выполнение этой операции в каждое воскресенье в 2.30.



При создании расписаний следует соблюдать некоторую осторожность. Если задача с расписанием содержит длительный процесс, то она может поставить другие процессы в очередь, а то и вообще не даст им выполниться. С резервным копированием такая проблема не возникает, однако, если это возможно, нужно постараться ставить задачи резервного копирования на разное время.

Щелкните на кнопке Next. Появится окно, показанное на рис. 11.20.

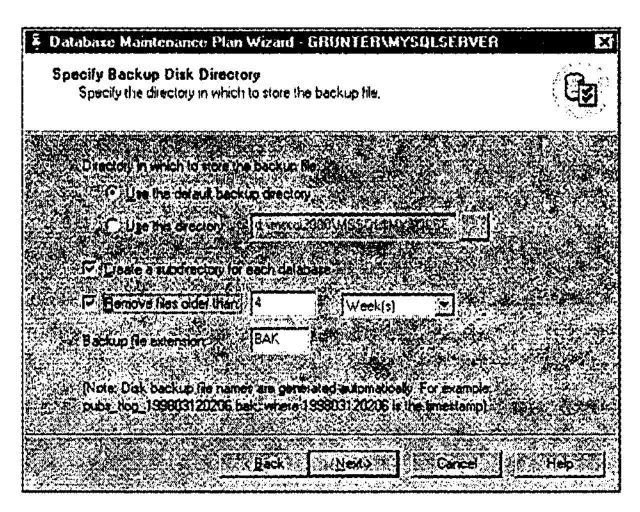


Рис. 11.20. Здесь определяются параметры хранения резервной копии

9. В этом окне устанавливаются описанные ниже опции.

- Directory in which to store the backup file (Каталог хранения файла резервной копии). Для этого можно использовать отдельный каталог или же указать каталог, задаваемый по умолчанию SQL Server 2000. Оставьте каталог по умолчанию.
- Create a subdirectory for each database (Создать подкаталог для каждой базы данных). Мы создаем резервные копии многих баз данных, поэтому будет удобнее расположить их в отдельных подкаталогах.
- Remove files older than (Удалять файлы, более старые, чем). Если этот флажок установлен, то SQL Server 2000 будет удалять файлы, созданные ранее заданного периода. Не устанавливайте этот флажок, если нужна подробная история базы данных. Сейчас установите значение 4 недели.
- Backup file extension (Расширение файла резервной копии). Оставьте значение опции по умолчанию — BAK. Можно задать другое расширение файла, однако опция по умолчанию вполне подходит.

На заметку В нижней части окна мастер информирует нас о том, что имя файла создается динамически. В состав имени входит "печать времени", которая представляет собой специальный тип данных, фиксирующий в имени файла дату и время его создания.

Еще раз щелкните на кнопке Next. Появится диалоговое окно Specify the Transaction Log Backup Plan (Создание плана резервного копирования журнала транзакций), как показано на рис. 11.21.

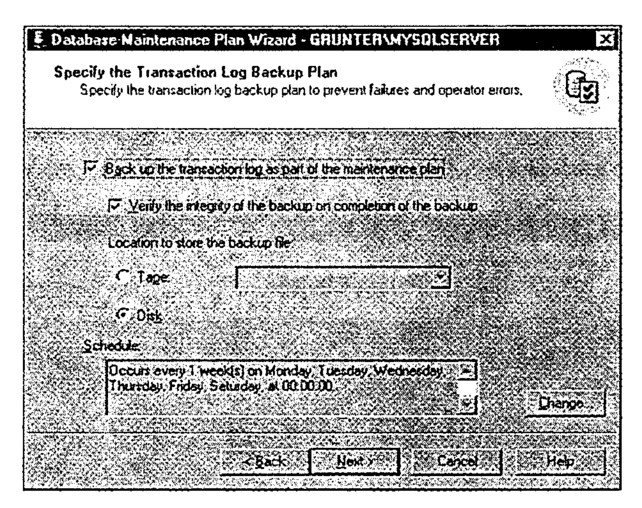


Рис. 11.21. Окно создания плана резервного копирования журнала транзакций

10. Резервные копии журнала транзакций будут сохраняться аналогично резервным копиям базы данных, а потому укажите в этом окне те же опции, кроме расписания. Задайте сохранение копий каждый день, кроме воскресенья, в 12 часов ночи (это расписание по умолчанию). Щелкните на кнопке Next.

11. Появляется диалоговое окно, аналогичное показанному на рис. 11.20. Задайте те же опции, что и в окне Specify backup disk directory, кроме расширения файла резервной копии, которое оставьте по умолчанию — TRN.

12. Щелкните на кнопке Next. Появится окно Reports to Generate (Создание отчетов), как показано на рис. 11.22.

13. В окне Reports to Generate можно задать автоматическое создание отчетов о шагах выполнения плана поддержки и возникших при этом ошибках. Если установить флажок Write report to a text file in directory (Записывать отчет в текстовый файл в каталог), то становится доступным ряд опций.

- Каталог, в котором сохраняется файл отчета.
- Delete text report files older than (Удалять текстовые файлы отчетов старше чем). Установите значение этого параметра равным 4 недели.
- Send e-mail report to operator (Отправить электронное письмо с отчетом оператору). Если раньше был успешно зарегистрирован оператор, то можно задать отправку ему отчета с помощью электронного письма.

Заполнив значения параметров этого окна, щелкните на кнопке Next. Это предпоследнее окно мастера.

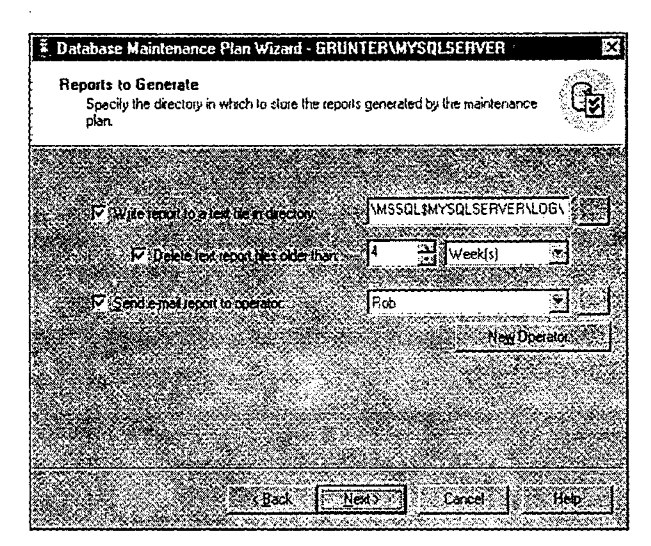


Рис. 11.22. Опции автоматического создания отчетов о выполненных действиях и возникших ошибках

14. В окне Maintenance History (рис. 11.23) можно задать сохранение информации о действиях сервера в ходе выполнения плана поддержки. Отчет об этих действиях заносится в таблицу.

Database Maintenance Plan Wizard - GRUNTER\MYSQLSERVER
Maintenance History
The record of the activities performed can be stored on the server on which they were performed, and/or another (remote) server.
Local server
Write history to the msdb.dbo.sysdbmaintplan_history table on this server
Limit rows in the table to: 1000 rows for this plan
Remote server
History is added to the msdb.dbo.sysdbmaintplan_history table on the remote server. Windows NT Authentication is used to log on to the remote server.
< Back
Next >
Cancel
Help

Рис. 11.23. Здесь устанавливаются опции протоколирования выполненных действий и возникших ошибок в системной таблице

В этом окне устанавливаются опции вставки строк в таблицу `sysdbmaintplan_history` системной базы данных `msdb`. Как и в случае отчета, записываются выполненные шаги, имя базы данных, успешное или неуспешное завершение и другая информация. Каждому выполненному действию отводится одна строка. Если одно и то же действие выполняется в разных базах данных, то для каждой из них выделяется отдельная строка.

Установите флажок Limit rows in the table to (Максимальное количество строк в таблице) и значение поля справа равным **1000**. Это значит, что если количество строк таблицы достигнет 1 000, то более старые строки будут удалены из таблицы.

Флажок Write history to the sever (Записывать историю на сервере) по умолчанию снят. При этом история записывается в текущем экземпляре SQL Server 2000. Однако можно задать запись истории на другом сервере или в другом экземпляре SQL Server 2000. Не устанавливайте этот флажок и щелкните на кнопке Next.

Это все! Осталось лишь задать имя плана поддержки в появившемся завершающем окне (рис. 11.24).

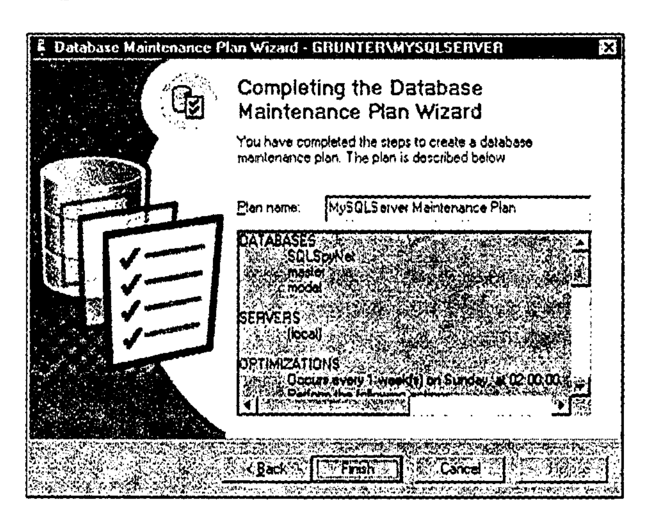


Рис. 11.24. Задание имени плана поддержки

15. В области просмотра этого окна можно увидеть установленные параметры конфигурации. Посмотрите их. Если что-то неправильно, то это последний шанс исправить ошибку (для этого щелкните на кнопке Back). Если вы остались всем довольны, то присвойте плану описательное имя (например, *MySQLServer Maintenance Plan*) и щелкните на кнопке Finish.

Дело сделано! Еще одна работа завершена успешно. Я уверен, вам все больше нравится роль настоящего администратора базы данных SQL Server.

Теперь в составе экземпляра SQL Server 2000 есть созданный план поддержки. Его можно увидеть в окне Enterprise Manager в папке Management\Database Maintenance Plan (рис. 11.25).

Созданный план обеспечит правильную и эффективную работу баз данных. В случае каких-либо аварий, создаваемые планом свежие резервные копии помогут восстановить базы данных.

Таким образом, администратор базы данных может пойти погулять в парке. Что еще осталось делать? Мы создали план, автоматически выполняющий типичные задачи администрирования, и предполагаем, что теперь все будет в порядке. Однако не заблуждайтесь на этот счет. От администратора базы данных требуется довольно высокая квалификация. Он должен уметь быстро решать разнообразные сложные проблемы, на него же возложена обязанность разработки тщательно продуманного плана действий в случае непредвиденных обстоятельств.

Рис. 11.25. Определенный в экземпляре SQL Server 2000 план поддержки базы данных

Поддержка индексов

В этой книге уже неоднократно упоминались индексы, однако до сих пор при разработке приложения не было примеров их использования или реализации.

При разработке базы данных без индексов не обойтись, однако большинство из них создаются автоматически, даже без ведома разработчика! Например, когда мы создаем первичный ключ, SQL Server 2000 автоматически создает уникальный индекс для столбца первичного ключа.

SQL Server 2000 поддерживает два типа индексов: *кластеризованные* и *некластеризованные*.

Термин

Если индекс *кластеризован*, то столбец, для которого он создан, хранится в упорядоченном виде. Например, если создать кластеризованный индекс для числового столбца, то строки будут храниться таким образом, что поля индексированного столбца будут расположены в порядке 1, 2, 3 и т.д. Этот тип индекса влечет за собой упорядочение данных, поэтому таблица может иметь только один кластеризованный индекс. Каждая вставленная строка располагается таким образом, что упорядоченность сохраняется. По умолчанию первичный ключ является кластеризованным индексом.



Термин

Некластеризованные индексы на диске не упорядочены, поэтому таблица может иметь сколько угодно таких индексов. Они содержат небольшие фрагменты данных и указатель на место хранения остальных данных на диске.

Однако это не объясняет, что такое индексы. Вы часто будете слышать (и я согласен с этим), что индекс таблицы базы данных напоминает предметный указатель в книге.

Предметный указатель в книге используется для быстрого поиска нужной информации. В SQL Server 2000 применяется тот же принцип. Если индекс есть, SQL Server 2000 использует его для поиска нужной информации. В то же время, если индекса нет, SQL Server 2000 сканирует таблицу, просматривая *каждую* строку, пока не встретит нужную. Разумеется, эта работа требует очень много времени и ресурсов, особенно если таблица большая.

Как работают индексы

Индекс содержит небольшую часть данных столбца и указатель на место диска, где хранятся эти данные таблицы. Например, если определен индекс для столбца `Firstname` таблицы `Person` и одно из полей столбца содержит `Jimmy the Rotten`, то, скорее всего, SQL Server поместит в индекс часть `Jimmy` (но, может быть, и все поле) и указатель на место диска, где хранится все поле (рис. 11.26).

Таблица Person

PersonID	Firstname	Surname	DOB	PhoneNo
1	Jimmy the Rotten	Bad Guy	NULL	Not Available
2	Greg	Cross	NULL	Not Available
3	Emma	Peel	NULL	Not Available
...	...	...	...	...
...	...	...	...	...

Индекс по столбцу Firstname

Emma
Greg
...
...
Jimmy

Рис. 11.26. Принцип создания индекса для столбца таблицы

Индекс содержит небольшую часть данных таблицы. Поэтому, если SQL Server должен найти информацию, содержащуюся в индексе, то он даже не обращается к таблице. При этом эффективность чтения данных значительно возрастает.

Можно ли назначить индекс для каждого столбца? Это позволило бы серверу быстро найти любые данные.

Все имеет свою цену. Хотя индексы и помогают ускорить чтение данных, потери все-таки неизбежны. Таблицы, содержащие индексы, занимают больше места на диске, потому что индексы являются объектами базы данных (как и все остальное), к тому же придется хранить копию данных для каждой строки.

Индексы существенно замедляют выполнение многих стандартных операций, таких как `INSERT`, `UPDATE` или `DELETE`. Это объясняется тем, что SQL Server 2000 должен постоянно поддерживать индексы. Если, например, вставляется 10 000 или более строк, то при каждой вставке приходится обновлять индекс, чтобы он правильно отражал состояние данных.

Когда следует использовать индексы

Как администратор базы данных, вы должны найти оптимальный баланс между числом индексов и частотой изменений данных, выполняемых пользователями. Эта

задача несколько облегчается тем, что SQL Server 2000 позволяет определять индексы для представлений. Это одно из совершенно новых средств SQL Server 2000. Индексы представлений работают аналогично индексам таблиц.

Однако SQL Server 2000 делает еще один шаг вперед. Если к представлению нет непосредственного обращения из предложения FROM оператора Transact-SQL, то программа оптимизации запросов может принять решение использовать индекс, созданный в представлении для чтения данных (если, конечно, запрос правильный!).

На заметку Индексы представлений (индексированные представления) не поддерживаются версией SQL Server 2000, которую мы сейчас используем. Чтобы воспользоваться этим новым средством, нужно установить Enterprise Edition SQL Server 2000.

Создание индекса для таблицы Person

Попытаемся создать индекс для таблицы Person, который ускорит выполнение запросов к ней. Правда, наше приложение пока очень маленькое, поэтому ускорение будет почти неощутимым. Однако в будущем, когда объем данных возрастет, ускорение станет весьма существенным.

Сначала выполним запрос к таблице без индекса. Но на этот раз, прежде чем реализовать индекс, воспользуемся еще одним инструментом отладки SQL Server 2000 — *планом выполнения*.



План выполнения — это графическое представление выполнения запроса сервером. Это отличное средство отладки сложных запросов, с его помощью легко обнаружить, где выполнение требует больших затрат процессорного времени.

В SQL Server 2000 поддерживаются два типа планов выполнения: оценочные и фактические.

Оценочные планы используются для анализа запроса, введенного в окне Query Analyzer, они дают предварительное представление о предстоящем выполнении запроса. Это полезно для больших запросов, когда требуется примерно оценить, как долго они будут выполняться.

Фактические планы выполнения создаются *после* выполнения запроса. Они представляют собой фактический результат реализации планов. Фактические планы содержат ценную информацию об узких местах выполнения запросов.

Проверка запроса без индекса с помощью плана выполнения

Ниже описано создание в Query Analyzer графического плана.

1. Выберите команду Query⇨Show Execution Plan (рис. 11.27) или нажмите комбинацию клавиш <Ctrl+K>.
2. Выбрав просмотр плана выполнения, введите код Transact-SQL, приведенный в листинге 11.2.

Рис. 11.27. Создание плана выполнения в Query Analyzer

Листинг 11.2. Выполнение оператора `SELECT` для таблицы `Person` без индекса

Код для запуска →

```
SELECT * FROM Person
WHERE Firstname LIKE '%a%'
```

Диалоговое окно Query Analyzer содержит вкладку Execution Plan. Активизируйте эту вкладку (рис. 11.28).



Если задержать указатель мыши на пиктограмме *Person.PK_Person*, то появится информация о выполнении сервером различных операций, возвращающих результат запроса.



Анализ

Сервер возвратил массу различной информации (может, даже слишком много). Сейчас нас интересует значение Physical Operation. Оно сообщает, что в запросе используется текущий кластеризованный индекс, определенный для таблицы (т.е. первичный ключ, который мы определили для таблицы).

Для нескольких строк, из которых состоит наша таблица, это может быть и хорошо, однако, если таблица содержит сотни строк, использовать для запроса первичный ключ нерационально. Ведь предложение `WHERE` применено к столбцу `Firstname`.

Создадим индекс для столбца `Firstname`; тогда, если таблица содержит сотни строк, программа оптимизации запросов назначит для чтения данных использование индекса.

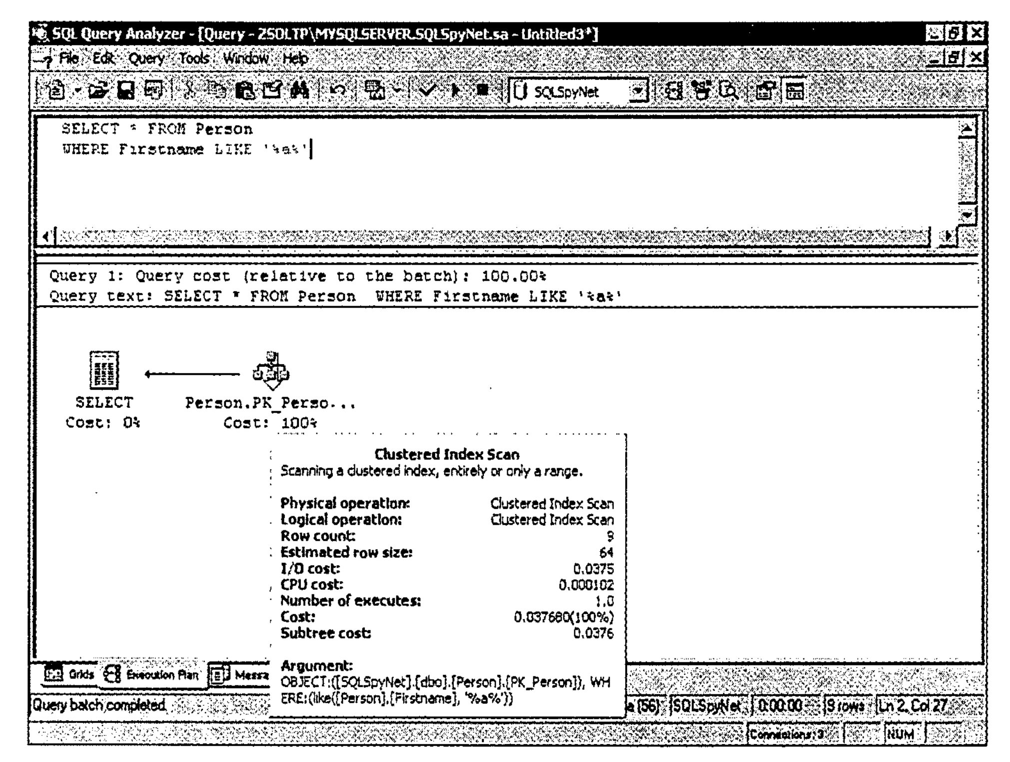


Рис. 11.28. Результаты выполнения запроса с включенным планом выполнения в uery Analyzer; в таблице присутствует кластеризованный индекс столбца `PersonID`

Индексация таблицы `Person`

Мы создаем некластеризованный индекс. Это связано с тем, что создать кластеризованный индекс мы не сможем, потому что таблица не может иметь больше одного кластеризованного индекса. Наша таблица уже имеет кластеризованный индекс, созданный сервером автоматически при создании первичного ключа `PersonID`.

Для создания индекса введите в Query Analyzer код Transact-SQL, приведенный в истинге 11.3.

Листинг 11.3. Создание индекса таблицы `Person` для ускорения выполнения запросов

Код для запуска

```
CREATE NONCLUSTERED INDEX idxPersonFirstname
ON Person (Firstname)
```

Теперь у нас есть первый индекс. Как все просто! Однако, прежде чем продолжить, рассмотрим структуру оператора создания индекса.



Анализ

В строке 1 определяется тип создаваемого индекса, в нашем примере это некластеризованный индекс с именем `idxPersonFirstname`.

В строке 2 указывается таблица (`Person`) и столбец (`Firstname`), для которых мы хотим создать индекс.

Проверка изменения эффективности после создания индекса

Еще раз выполним предыдущий запрос, однако теперь внесем небольшое изменение (листинг 11.4).

Листинг 11.4. Выполнение оператора `SELECT` с индексом `idxPersonFirstname`, определенным для таблицы `Person`; задано обязательное использование индекса

```
1: SELECT * FROM Person
2: WITH (INDEX(idxPersonFirstname))
3: WHERE Firstname LIKE '%a%'
```

Анализ

В строке 2 указывается, что SQL Server должен использовать новый индекс. Это необходимо сделать, иначе программа оптимизации запросов отменит использование индекса, потому что при таком небольшом объеме данных индекс бесполезен.

На заметку

Заставлять SQL Server 2000 использовать при чтении данных индекс — признак плохого тона. Программа оптимизации запросов понимает внутренние процессы, протекающие в сервере, значительно лучше, чем мы. Сейчас мы сделали это только для демонстрации работы индекса.

Посмотрите теперь на план выполнения (рис. 11.29). Как видите, SQL Server 2000 при выполнении запроса использовал новый индекс.

Поместите указатель мыши над пиктограммой `Person.idxFirstname`. Вы увидите, что для чтения данных из таблицы SQL Server был вынужден использовать индекс.

Итак, мы создали индекс и посмотрели, как он используется. Теперь рассмотрим управление индексами с целью оптимизации выполнения запросов.

Поддержка индексов для обеспечения эффективной работы приложений

Индексы способны повысить эффективность работы приложений, однако, как и се остальное в базах данных, они нуждаются в поддержке. Особенно если в базе данных интенсивно выполняются транзакции, а количество изменений данных доходит до сотен тысяч в день.

Индексы можно удалять, изменять или создавать новые, не изменяя схему базы данных. Индексы всего лишь увеличивают скорость выполнения, поэтому эксперименты с ними (конечно, в определенных рамках) не могут повредить базу данных.

SQL Query Analyzer - [Query - ZSOLTP\MYSQLSERVER.SQLSpyNet.sa - Untitled1*]

```
SELECT * FROM Person
WITH (INDEX(idxPersonFirstname))
WHERE Firstname LIKE '%a%'
```

```
Query 1: Query cost (relative to the batch): 100.00%
Query text: SELECT * FROM Person  WITH (INDEX(idxPersonFirstname))  WHERE Firstname LIKE '%a%'
```

SELECT Cost: 0% ← Bookmark Lookup Cost: 14% ← Person.idxPerso... Cost: 8

Index Scan
Scanning a non-clustered index, entirely or only a range.

Physical operation:	Index Scan
Logical operation:	Index Scan
Row count:	9
Estimated row size:	36
I/O cost:	0.0375
CPU cost:	0.000102
Number of executes:	1.0
Cost:	0.037680(86%)
Subtree cost:	0.0376

Argument:
OBJECT:([SQLSpyNet].[dbo].[Person].[idxPersonFirstname]), WHERE:(like([Person].[Firstname], '%a%')), FORCEDINDEX

Grids | Execution Plan | Messages

Query batch completed. 0:00:00 | 9 rows | Connections: 3 | NUM

Рис. 11.29. План выполнения запроса с использованием некластеризованного индекса

Однако раз уж индексы таблиц и представлений созданы, необходимо периодически проверять эффективность их использования.

Существует два способа обновления индексов.

Индексы можно удалять и создавать заново. Однако это может отрицательно повлиять на эффективность работы системы, потому что придется удалять и создавать их несколько раз.

Рассмотрим это на примере таблицы Person. Для нее определены два индекса: кластеризованный и некластеризованный. При удалении кластеризованного индекса удаляется и создается заново некластеризованный индекс, потому что его указатели ссылаются на таблицу кластеризованного индекса (если он есть). При создании заново кластеризованного индекса некластеризованный опять удаляется и создается заново.

Этого можно избежать, если указать серверу, что мы хотим удалить и создать индекс за один шаг. Тогда SQL Server 2000 будет учитывать, что мы только перестраиваем индекс, а не удаляем и создаем его.

Второй способ перестройки индексов таблицы — использование оператора DBCC DBREINDEX. Это наиболее легкий способ обновления статистики индекса. Рассмотрим его использование на конкретном примере (листинг 11.5).

Листинг 11.5. Выполнение процедуры DBCC, обновляющей индексы таблицы Person




```
1: DBCC DBREINDEX(Person)
```

С помощью оператора DBCC DBREINDEX можно обновить как все, так и один индекс таблицы.

На заметку Для обновления одного индекса нужно указать его имя после имени таблицы:

```
DBCC REINDEX(Person, PK_Person)
```

Этот оператор обновит только индекс PK_Person таблицы Person.

Теперь SQL Server 2000 обновил все индексы таблицы Person, как показано на рис. 11.30.

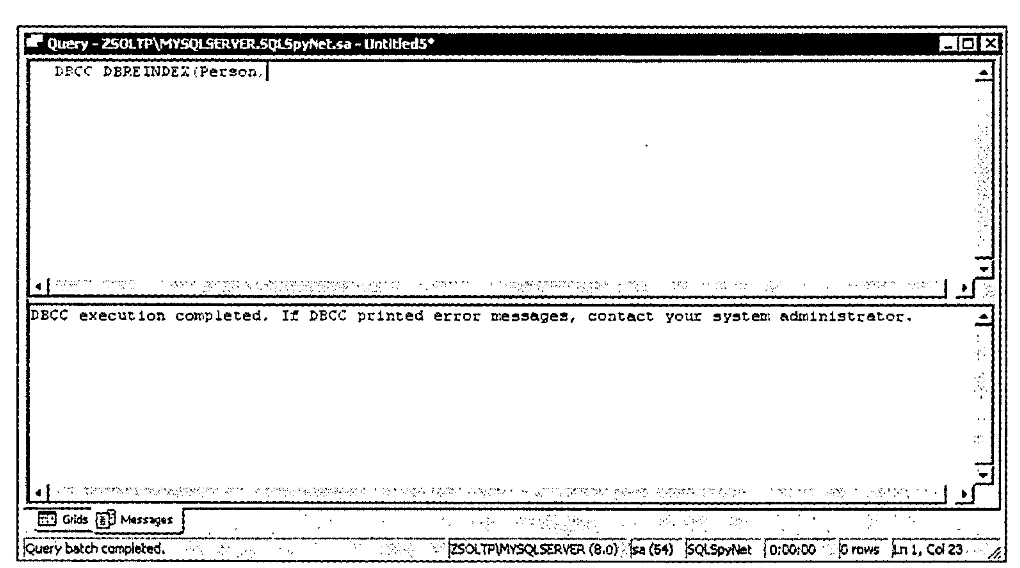


Рис. 11.30. Окно плана выполнения в Query Analyzer, перестройка индексов таблицы Person

Все это очень хорошо, но самое лучшее в том, что вам не придется выполнять эту операцию часто, возможно, даже никогда!

Как и следовало ожидать, SQL Server 2000 периодически обновляет статистику индексов самостоятельно. Весьма любезно, не правда ли? За исключением случая, когда при создании индекса вы явно указали, что не хотите, чтобы SQL Server 2000 обновлял статистику. Но делать так не рекомендуется.

А сейчас перейдем к самому интересному в работе администратора базы данных — управлению производительностью.

Управление производительностью

Наиболее увлекательная часть работы администратора баз данных (по крайней мере, мне так кажется) — управление производительностью. Средства управления производительностью позволяют посмотреть, что происходит внутри сервера, а затем принять надлежащие меры для устранения обнаруженных неполадок. Это звучит довольно просто, однако подобная задача достаточно сложна, особенно в изменяющейся среде сети или в системе, выполняющей сотни тысяч транзакций.

Как управлять работой сервера? Откройте папку Management в окне Enterprise Manager. В ней есть папка Current Activity (Текущая активность).

В этой папке можно увидеть, что каждый пользователь делает в каждой из ваших баз данных и, более того, с каждой таблицей. Можно подойти с обратной стороны и посмотреть, какие операции выполнялись с каждой таблицей и какими пользователями.

Так что давайте подробно ознакомимся с объектами папки Current Activity.

Управление текущей активностью

Как уже отмечалось, в папке Current Activity можно увидеть, что происходит на сервере. Папка активности содержит три основных объекта.

- Process Information — просмотр текущей активности на сервере.
- Locks/Process ID — просмотр действий клиентов на сервере.
- Locks/Objects — просмотр действий базы данных на сервере.

Как выглядит выполняющийся процесс? С помощью цикла запустим длительный процесс, чтобы можно было наблюдать его в окне.

Откройте окно Query Analyzer, воспользовавшись для этого регистрационным именем `SQLSpyNetUser`. Введите текст листинга 11.6.

Листинг 11.6. Выполнение цикла для анализа информации о пользователе

Код для запуска

```
DECLARE @Counter INT
SET @Counter = 1
WHILE @Counter < 1000000
   BEGIN
      PRINT 'Counter is '+CONVERT(VARCHAR(10), @Counter)
      SET @Counter = @Counter + 1
   END
```

Выполняя эту программу, SQL Server 2000 будет вынужден повторить тело цикла 999 999 раз! Пока он будет это делать (что займет немало времени), мы сможем проанализировать его работу.

Теперь в папке Management\Current Activity\Locks/Objects посмотрите параметр SQLSpyNet; вы увидите значение SPID текущего пользователя. Если дважды щелкнуть на значении SPID, то можно увидеть команду, выполняемую пользователем (рис. 11.31).

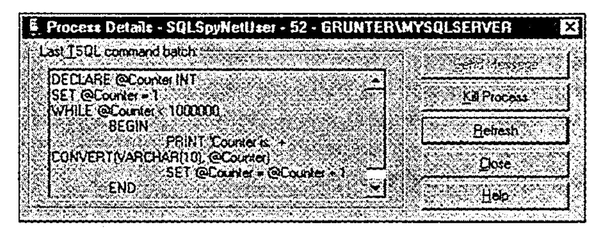


Рис. 11.31. Подробности процесса пользователя `SQLSpyNetUser`, выполняющего команду в базе данных SQLSpyNet

Термин *SPID* — это уникальный системный идентификатор (ID) процесса, присвоенный соединению каждого пользователя. Его значение не является постоянным числом.

Один из мощнейших инструментов администратора базы данных — команда KILL. С ее помощью можно "остановить" пользователя. Эта команда останавливает процесс пользователя и отменяет его команду.

Если пользователь выполняет длительный процесс (как мы сейчас), то можно выполнить команду KILL или щелкнуть на кнопке Kill Process. Этим будет прерван процесс пользователя и задана команда отмены его транзакции. Выполняемые пользователем изменения при этом отменяются.



Если процесс пользователя должен выполняться 45 минут, а вы прервали его на 40-й минуте, то серверу потребуется еще 40 минут, чтобы отменить сделанные пользователем изменения.

Команду KILL можно выполнить с помощью Transact-SQL, но для этого нужно знать SPID пользователя.

Теперь у вас есть общее представление о средствах контроля активности, предоставляемых SQL Server 2000. Однако это только самое общее представление. Если вы хотите узнать больше, запустите предыдущую программу Transact-SQL с длинным циклом и посмотрите другие папки в Current Activity. Там вы увидите, какие процессы выполняются и как они влияют на сервер.

Резюме

Мы рассмотрели основы задач администрирования баз данных SQL Server 2000. Заложен фундамент знаний, которые понадобятся вам, как администратору баз данных, в самом ближайшем будущем.

Рассмотренные вопросы не отражают всего, что должен знать администратор баз данных, но дают представление о том, что ожидает вас в этой роли. И это оказалось весьма интересным, не правда ли?

Теперь, используя полученные знания в качестве основы, вы сможете самостоятельно администрировать экземпляр SQL Server 2000 и настраивать его по своему вкусу. Успехов вам в этом деле!

Следующие шаги

В следующей главе мы опять "сменим шляпу" и рассмотрим разработку интерфейса пользователя. В конце концов, мы не можем потребовать от пользователя, чтобы он знал SQL не хуже нас. Используя новейшие технологии компании Microsoft (а чьи же еще?) мы создадим Web-интерфейс, с помощью которого можно будет передать данные броузеру. Итак, приступаем!

ГЛАВА 12

Разработка интерфейса пользователя базы данных SQLSpyNet

В этой главе...

И вновь вернемся к роли разработчика баз данных, но теперь займемся программированием.

Итак, у нас есть база данных, удовлетворяющая наши потребности. Структура базы данных стабильна, существует план резервного копирования, поддерживающий систему доступа 24 часа в сутки в течение 7 дней в неделю, реализована основа поддержки многих пользователей. Однако далеко не все наши пользователи (кроме нас самих, конечно) обладают достаточными знаниями языка Transact-SQL для выполнения повседневных задач.

Как можно помочь им? Используя знания в области программирования, мы должны создать дружественный и простой для понимания интерфейс пользователя, с помощью которого они смогут легко и быстро вводить и считывать данные. Интерфейс пользователя должен быть интуитивно понятным и безопасным, в нем не должно быть трудноразрешимых ситуаций.

Выполнение этих требований кажется не таким уж сложным, однако неопытный программист может потратить не один месяц на освоение тонкостей языка программирования, не говоря уже о создании полнофункциональной системы.

К тому же разработчик должен решить, какая технология более всего подходит для разработки данного приложения и какую архитектуру лучше использовать при его развертывании.

К счастью, вы меня понимаете! Двигайтесь вперед, и, не успев опомниться, вы создадите фундамент Web-интерфейса пользователя, на основе которого сможете самостоятельно наращивать функциональные возможности приложения.

Сначала кратко рассмотрим архитектуры, используемые при разработке приложений.

Основы архитектуры клиент/сервер

Хорошо разработанная архитектура — ключевое звено успеха всего приложения. Если архитектура плохо продумана, то система и функциональные возможности приложения не могут быть расширены на более поздней стадии.

Если раньше вам приходилось программировать, то у вас должно быть общее представление об архитектуре клиент/сервер. Мы коротко останавливались на этом при установке SQL Server 2000. Чтобы освежить знания в этом вопросе, обратитесь к приложению Б, "Установка и настройка SQL Server 2000".

Архитектура клиент/сервер — это нечто большее, чем просто выполняемые ими роли. Существуют также функциональные средства различных частей. И здесь появляется концепция интеллектуального и/или тонкого клиента.

Архитектура клиент/сервер также описывает концепцию логических уровней внутри архитектуры (рис. 12.1). Эти уровни инкапсулируют функциональные области, выполняющие различные задачи.

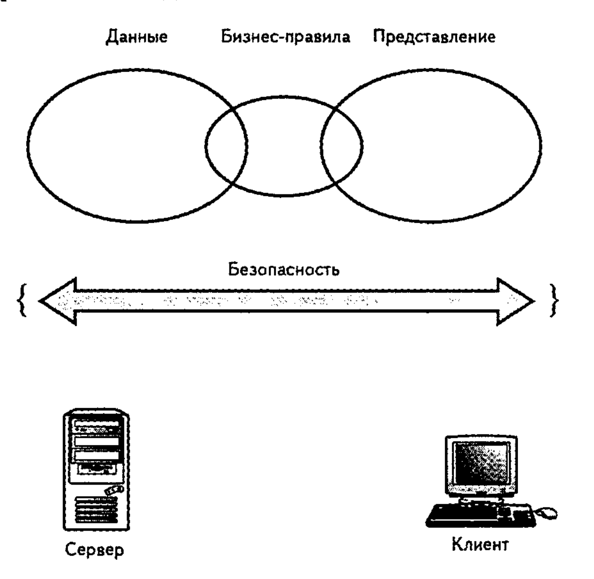


Рис. 12.1. Уровни архитектуры клиент/сервер в пространстве между серверной частью и интерфейсом пользователя



Уровень — это отчетливо различимая функциональная область внутри архитектуры; однако уровень не существует сам по себе. Он тесно взаимодействует с другими уровнями, создавая таким образом необходимые функциональные средства нашего приложения.

- Уровень представления. Это графический интерфейс, предоставленный пользователю. Уровень представления содержит программу, форматирующую данные и представляющую их в легком для чтения виде. Данные, поступившие со входа пользователя, передаются уровню бизнес-правил.
- Уровень бизнес-правил. Содержит бизнес-правила приложения. Например, если некто должен быть старше 16 лет, то уровень бизнес-правил проверяет,

так ли это. В случае положительного решения уровень бизнес-правил передает данные уровню данных, в противном случае возвращает сообщение об ошибке уровню представления.

- Уровень данных. Считывает проверенные данные из уровня бизнес-правил, затем пытается вставить их в базу данных; но, если необходимое значение ключевого поля отсутствует, он возвращает данные уровню представления.

Вооружившись этими знаниями, кратко рассмотрим архитектуру с интеллектуальными клиентами.

Интеллектуальный клиент

Интеллектуальный клиент (иногда используется термин *двухуровневый клиент/сервер*) — это место взаимодействия хорошо разработанного интерфейса пользователя с серверной частью базы данных. Такие приложения, как Microsoft Access, могут выполняться на локальном компьютере и взаимодействовать с сервером, содержащим базу данных SQL Server 2000. Интерфейс пользователя этого типа содержит много встроенных уровней архитектуры клиент/сервер, включая уровень представления, уровень бизнес-правил и даже часть уровня данных (рис. 12.2).

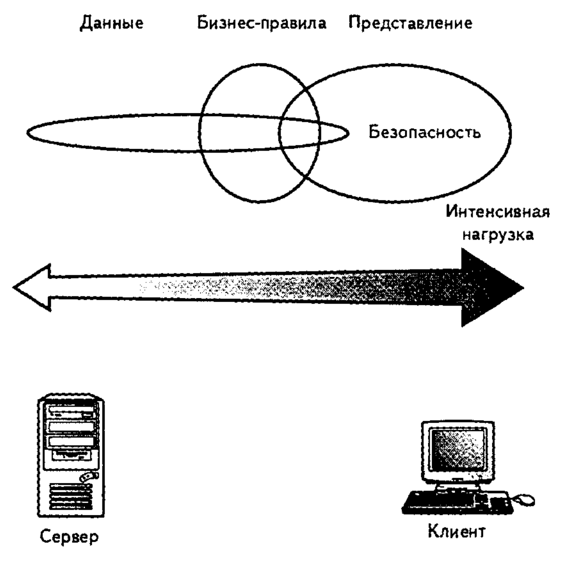


Рис. 12.2. В пользовательской части интеллектуальный клиент берет на себя основную нагрузку

Такой тип приложения содержит основную часть выполняемых логических и вычислительных процедур. Это позволяет быстро проверять входные данные и возвращать сообщения об ошибках, не обращаясь к серверу.

Архитектура с интеллектуальными клиентами обладает следующими реальными преимуществами:

- позволяет быстро проверять правильность данных;
- облегчает управление безопасностью данных;
- в общем случае облегчает и ускоряет разработку приложений;

- уменьшает нагрузку на сеть;
- использует возможности компонентов операционной системы;
- позволяет использовать недорогой сервер;
- если сервер терпит крах, локальное приложение может остаться доступным.

Однако все же архитектура этого типа не идеальна; ей присущи такие недостатки:

- если возникает потребность изменить программу, то ее нужно повторно устанавливать на каждом компьютере;
- велики затраты в клиентской части, потому что интеллектуальный клиент требует значительных вычислительных ресурсов;
- ее трудно поддерживать, особенно если приложение устанавливается на компьютерах многих клиентов;
- стоимость поддержки может значительно увеличиться.

Это не все "за" и "против" архитектуры с интеллектуальным клиентом, однако они довольно полно показывают, с чем столкнется разработчик при выборе оптимальной архитектуры.

Теперь рассмотрим архитектуру с тонким клиентом.

Тонкий клиент

Тонкий клиент (иногда используется термин *многоуровневый клиент/сервер*) позволяет провести более четкую границу между уровнями приложения.

На заметку Сейчас под тонким клиентом все чаще подразумеваются Web-приложения, получившие широкое распространение.

Этот тип архитектуры предполагает расположение уровня представления на клиентском компьютере, уровня бизнес-правил — на центральном сервере, а уровня данных — на сервере баз данных SQL Server (рис. 12.3).

Как вы заметили, архитектура с тонкими клиентами выглядит более структурированной, чем с интеллектуальными. В ней каждый уровень логически четко отделен от другого. Однако, несмотря на то что уровни четко разделены, в процессе работы приложения они интенсивно взаимодействуют и существенно зависят друг от друга.

Архитектура с тонкими клиентами имеет следующие преимущества:

- если нужно изменить программу, то изменение осуществляется только в одном месте;
- эту систему легко установить;
- ее легко поддерживать, потому что все программы находятся в одном месте;
- уровень представления выполняется только на компьютере клиента, поэтому требования к его ресурсам невелики;
- она гармонично вписывается в модель клиент/сервер.

Таков краткий обзор архитектуры с тонкими клиентами. И хотя он далеко не исчерпывающий, но дает достаточное представление для начала работы с базами данных.

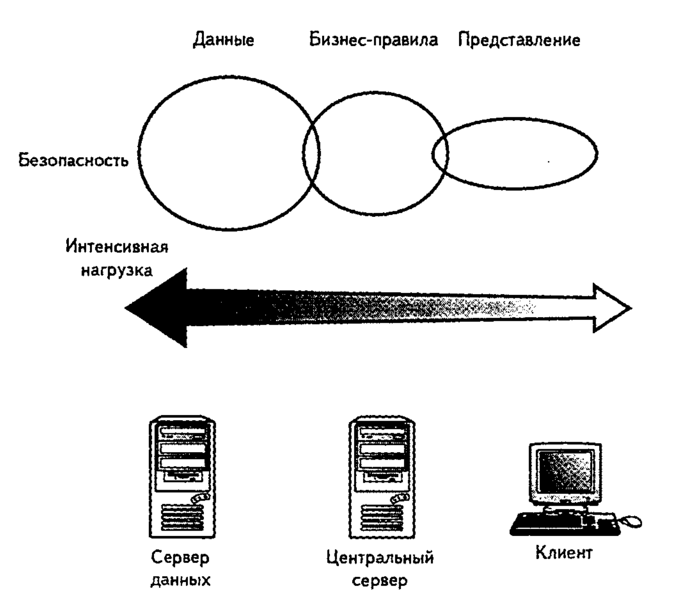


Рис. 12.3. Использование центрального сервера с целью удаления источника данных от клиентской части

Какая конфигурация лучше

Все зависит от характера решаемых задач. Если число пользователей превышает 200, причем у всех разные компьютеры, а у некоторых даже устаревшие с 486-м процессором, то, скорее всего, лучше подойдет архитектура с тонкими клиентами. Но если количество пользователей не превышает 50, пропускная способность сети невелика, а все локальные компьютеры самые современные, то следует использовать архитектуру с интеллектуальными клиентами.

Выбор типа архитектуры зависит от многих факторов. Один из них — предыдущий опыт программистов. Если программисты не имеют опыта создания Web-узлов, то в архитектуре с тонкими клиентами они будут разрабатывать приложение намного дольше, чем в архитектуре с интеллектуальными клиентами.

В вопросе выбора типа архитектуры не существует однозначных ответов. Если для решения какой-либо задачи была выбрана архитектура с тонкими клиентами, то при повторении подобной ситуации совсем не обязательно возвращаться к этому выбору. Если решение удовлетворяет требования клиентов, может быть реализовано в заданный срок и в пределах выделенных средств, то чего еще можно желать?

Выбор среды разработки интерфейса пользователя

Теперь вы имеете представление об архитектуре клиент/сервер как с интеллектуальными, так и с тонкими клиентами. Далее рассмотрим, какую среду разработки можно использовать для создания интерфейса пользователя базы данных SQLSpyNet.

На заметку Возможно, вы уже догадались, что я страстный фанат продуктов Microsoft. Мне нравятся инструменты разработки, предоставляемые этой компанией. Я получаю истинное наслаждение от работы с ними. Однако, это вовсе не значит, что программные продукты других компаний плохие. Тем не менее в дальнейших разделах при разработке интерфейса пользователя мы сосредоточимся на инструментах именно компании Microsoft.

Microsoft Visual Basic

Среда Visual Basic 6.0 содержит средства разработки приложений в любом из типов архитектуры клиент/сервер. С ее помощью можно создавать любые виды приложений, а ее будущие версии, конечно же, предоставят еще больше гибкости.

На заметку Недавно компания Microsoft объявила, что новая версия Visual Basic 7.0 будет основана на новой платформе .Net. Это значительно расширит функциональные возможности Visual Basic, предоставляя таким образом пользователям все средства, имеющиеся в Delphi и Java. Объектно-ориентированная платформа VB.Net будет поддерживать наследование, полиморфизм и инкапсуляцию. Я ожидаю новую версию с большим нетерпением.

Среда разработки значительно облегчает создание приложения. Она содержит инструменты разработки графического интерфейса пользователя, с помощью которых можно создавать формы и заполнять их, перетаскивая компоненты с панели инструментов.

Язык программирования основан на исходном языке Basic, существенно усовершенствованном. Больше нет оператора GOTO (хоть он по-прежнему поддерживается). Функции, процедуры и другие средства языка теперь намного проще понимать и использовать. Большинство программистов легко освоят этот язык.

С помощью Visual Basic 6.0 можно создавать весьма совершенные формы с окнами сообщений и элементами управления, что предоставляет все возможности для создания интеллектуального клиента. Используя Visual Basic 6.0, можно также организовать Web-классы, позволяющие передавать данные броузеру, создавая таким образом тонкого клиента.

Итак, среда Visual Basic 6.0 — весьма совершенный инструмент разработки. Правда, может возникнуть вопрос: неужели нет никаких "но"? По-видимому, нет.

Единственное, что препятствует еще более широкому распространению Visual Basic, — высокая начальная цена программного обеспечения. По сравнению с другими поставляемыми Microsoft средами разработки (например, Access или FrontPage) Visual Basic относительно дорог. Но он того стоит!

Тем не менее в этой главе мы не будем использовать Visual Basic для создания интерфейса, потому что, на мой взгляд, не каждый читатель этой книги знаком с ним.

Microsoft Access

Далее рассмотрим Microsoft Access 2000, входящий в состав пакета Microsoft Office 2000. Этот инструмент предназначен главным образом для начинающих, однако опытные разработчики также оценят богатство его среды разработки. В качестве языка программирования в Microsoft Access 2000 используется VBA (Visual Basic for Applications).

На заметку Мы будем говорить об Access 2000, хотя SQL Server 2000 поддерживает любую версию Access, даже самую первую. Однако Access 2000 содержит дополнительные функциональные возможности, которые отсутствуют в предыдущих версиях. Поэтому рекомендуем переходить на нее без колебаний.

Язык VBA — это "урезанная" версия Visual Basic. Он выглядит и работает почти так же, как и хорошо знакомый Visual Basic, однако в нем нет всего богатства возможностей этого языка. Но с каждым новым выпуском разница между ними становится все менее заметной.

Microsoft Access — это настольная база данных. С ее помощью удобно разрабатывать относительно простые и недорогие базы данных. Пользовательский интерфейс Microsoft Access довольно развит, он позволяет хранить данные локально в таблицах, непосредственно связанных с интерфейсом. Более того, среду Access можно связать с SQL Server 2000, выбрать из него информацию в локальную базу данных, обработать эту информацию и обновить данные в SQL Server.

Термин

Настольные базы данных используются для локального хранения информации в компьютере. С их помощью можно создавать таблицы, поддерживать ссылочную целостность и выполнять запросы. Обычно настольная база данных предоставляет возможность создать интерфейс пользователя, применяющий формы и элементы управления Windows. Компания Microsoft поставляет две настольные базы данных: Access и Visual FoxPro. Они имеют примерно одинаковые функциональные возможности, однако в некоторых частях света (по крайней мере в Новой Зеландии) Access распространена значительно шире.

Основной недостаток Access состоит в том, что это не полновесная база данных. Она хороша главным образом в архитектуре с интеллектуальными клиентами. В Access встроены формы и Web-страницы, но пользоваться ими пока довольно трудно.

На заметку

Какая разница между Access 2000 и SQL Server 2000 Personal Edition? SQL Server 2000 Personal Edition предоставляет меньшие возможности, чем коммерческое издание, тем не менее это полноценная среда разработки баз данных. В то же время Access — это гибрид Visual Basic и SQL Server с урезанными возможностями их обоих.

С помощью Access можно создавать структуру базы данных, однако ее широкое коммерческое распространение вряд ли целесообразно (кроме тех случаев, когда пользователей всего несколько), потому что Access не является полномасштабной средой разработки баз данных.

Планируя эту книгу, я обдумывал возможность разработки интерфейса пользователя в Access, но отказался от этого, поскольку вряд ли каждый читатель имеет Access на своем компьютере.

Итак, приступаем к разработке пользовательского интерфейса для Web. Интересно? Я так и предполагал.

Active Server Pages

Active Server Pages (ASP) — это последняя инициатива Microsoft, позволяющая динамически создавать страницы HTML на основе информации, хранящейся в базе данных



Экскурс

Немного о буквах

В современном мире трехбуквенные сокращения встречаются на каждом шагу, поэтому практически невозможно избежать дублирования их значений. Это же относится и к ASP.

Вот первое значение этой аббревиатуры: Active Server Pages. Это динамические Web-страницы, и сейчас мы приступим к их более подробному изучению.

Экскурс

Второе значение — Application Service Providers. Это компании, реализующие программные продукты. Например, некоторая компания купила лицензию на Microsoft Office и установила его на своем компьютере. Однако, когда выходит следующая версия Office, компания вынуждена опять покупать лицензию. Чтобы избежать этого, можно заключить соглашение с ASP-компанией и выплачивать ей ежемесячную ренту, за что она будет периодически обновлять программный продукт. Для небольших и средних компаний такая услуга незаменима.

Технология ASP обеспечивает Web-страницы данными, которые заполнили бы сотни статических страниц HTML.

Страницы ASP выполняются на Web-сервере IIS (Internet Information Server) или PWS (Personal Web Server). Броузеру ASP-страница передается в виде HTML-потока (текст). Это значит, что броузер может интерпретировать страницу и отображать ее. А еще говорят, что Microsoft — монополист! Однако Web великолепен, не так ли?

Технология ASP использует язык VBScript (или JavaScript). VBScript — это "сценарная" версия Visual Basic. Набор средств VBScript даже еще более ограничен, чем VBA, однако программирование от этого становится только интереснее.

В VBScript отсутствует типы данных. Все данные имеют тип `Variant` (специальный тип данных, способный содержать все типы), аналогично типу `Variant` в Visual Basic. Когда переменной первый раз присваивается значение (например, строка), в дальнейшем все значения этой переменной интерпретируются как строки. Аналогичное утверждение справедливо для целых и для всех других типов, поэтому, присваивая значения, нужно соблюдать некоторую осторожность. Это будет гораздо понятнее, когда мы начнем писать программу.

Технология ASP выглядит довольно привлекательно. Нужно ли чего-либо остерегаться? Конечно! ASP — великолепная технология, но следует понимать: все, что может быть передано броузеру, — это текст HTML. Контролировать входные данные пользователя непросто. Это можно делать с помощью тщательно продуманного *сценария на стороне клиента*, однако не все броузеры поддерживают языки сценариев.

Термин

В последнее время в области разработки Web и компонентов слово *сценарий* встречается очень часто. Особенно в контексте ASP, Windows Scripting Host (WHS) и клиентских сценариев. В терминологии броузеров клиентские сценарии подразумевают окна сообщений и процедуры проверки данных, выполняемые на Web-узле. Вам придется еще чуть-чуть побыть в неведении, прежде чем вы поймете, о чем речь.

Пока страница не выполняется, она не компилируется. Следовательно, если на странице есть ошибка, то она не может быть обнаружена, пока страницу не запросит броузер.

А как насчет среды разработки? Вот где проявляется мощь ASP! Если у вас есть Visual InterDev (из поставки Visual Studio), то вы во всеоружии. Если же у вас нет доступа к этим чудесным инструментам, вы можете писать код ASP с помощью обычного текстового редактора, подобного Блокноту.

На заметку

Среда Visual InterDev — великолепное средство разработки приложений для Web! В ней можно перетаскивать компоненты, писать программы и даже устанавливать соединение с базой данных, включив таким образом все данные, таблицы и структуры в среду разработки. С помощью Visual InterDev можно создавать хранимые процедуры, выполнять их и просматривать результаты, даже не запуская соответствующей утилиты SQL Server 2000! Среда Visual InterDev содержит такие средства разработки, как точки останова, окна наблюдений и др. Если у вас есть шанс воспользоваться этими возможностями — не раздумывайте!

Язык ASP известен как язык сценариев, а страницы HTML/ASP — это только текст, поэтому все приложение можно разработать в Блокноте или в любом другом текстовом редакторе, например WordPad или MS Word. Это довольно трудоемкая работа, однако таким образом можно по крайней мере начать разработку.

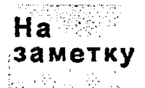


Для просмотра страниц ASP в Notepad или в любом другом текстовом редакторе нужно сначала его запустить, а затем выбрать команду *File⇨Open* или перетащить ASP-файл в открытое окно редактора.

Можно также добавить расширение .ASP в состав открываемых с помощью Notepad документов. Более подробную информацию об этой операции можно найти в справочной системе Windows.

Мы будем разрабатывать ASP-страницы с помощью программы FrontPage. Ее упрощенная версия поставляется с Internet Explorer 5.0, который у вас конечно же есть, иначе вы не смогли бы запустить SQL Server 2000. Убедитесь, что программа FrontPage у вас установлена. Если нет, то ее можно загрузить с Web-узла Microsoft.

Единственное, что еще должно быть установлено на вашем компьютере, — это Web-сервер. Для Windows 95/98 и Windows NT 4.0 Workstation должен быть установлен PWS (Personal Web Server). Для Windows 2000 (всех версий) и Windows NT 4.0 Server должен быть установлен IIS (Internet Information Server).



PWS можно установить при инсталляции Windows 98. В приложении А, "Установка Web-сервера", приведены сведения по установке и конфигурированию компьютера в качестве Web-сервера.

Установка соединения с базой данных SQLSpyNet

Прежде чем углубиться в разработку интерфейса пользователя, кратко рассмотрим способы установки соединения с нашей базой данных. Существует два подхода к установке соединений.

Использование испытанного методаODBC (MSDASQL)

ODBC (Open Database Connectivity) — стандартный метод установки соединения с базой данных. Он используется большинством приложений клиент/сервер, потому что драйверы ODBC легко конфигурируются в панели управления Windows. После этого достаточно указать в приложении клиента имя источника данных (Data Source Name — DSN), и соединение готово!

Для переопределения соединения при переносе сервера нет необходимости менять исходный текст программы. Достаточно открыть панель управления Windows и отредактировать свойства соединения.

Драйверы ODBC работают в качестве интерпретаторов разных типов данных. Они созданы для SQL Server, Access (Jet), Oracle, DB2, Excel, текстовых файлов и пр. При подаче запроса к базе данных с помощью ODBC этот запрос интерпретируется в код SQL запрашиваемой базы данных.

Конечно, это замедляет скорость выполнения запроса. За простоту поддержки приходится платить эффективностью выполнения. И плата эта довольно высока. Однако непрерывно создаются все лучшие драйверы ODBC, поэтому в будущем снижение производительности окажется, по-видимому, не столь ощутимым.

На заметку Microsoft рекомендует использовать ODBC только для совместимости с предыдущими версиями, например для установки соединения с ранее разработанными приложениями. В настоящее время для установки соединения с базой данных рекомендуется использовать новый метод OLEDB.

Использование новейшего провайдера OLEDB для SQL Server (SQLOLEDB)

Чем можно заменить ODBC? Средства SQL Server 2000 (и более ранних версий) позволяют устанавливать соединение с другим SQL Server 2000 непосредственно с помощью собственного провайдера SQLOLEDB. Можно язык сломать, не так ли? Этот метод создает соединение с базой данных, предоставляя некоторые строковые параметры соединения, почти как в ODBC. Самый большой недостаток SQLOLEDB проявляется при переносе сервера. Строковая информация о соединении определена в приложениях клиентов, поэтому, чтобы заставить приложения обращаться к новому серверу, нужно иметь доступ не только к центральному DSN, но и к исходным текстам программ клиентов.

Использование SQLOLEDB обеспечивает ряд возможностей, не поддерживаемых соединением ODBC. Оно предоставляет большую гибкость, а эффективность выполнения великолепна! Поэтому в нашем приложении будем использовать SQLOLEDB.

На заметку Для ASP-приложений перенос сервера при использовании SQLOLEDB не является проблемой. Строковая информация о соединении хранится в одном месте, а поскольку ASP — это только текстовый файл, информацию о соединении с сервером очень просто изменить.

Создание пользовательского интерфейса SQLSpyNet

В этом разделе рассматривается построение основы интерфейса пользователя, который устанавливает соединение с базой данных, подтверждает регистрационное имя пользователя и считывает (ищет) данные. Это только начало разработки приложения, однако оно дает необходимые строительные блоки, с помощью которых вы сможете развивать приложение самостоятельно.

Эта глава, в отличие от других, во многом напоминает рецепт: планирование страниц, создание учетной записи пользователя для доступа к страницам и т.д. Придерживайтесь этого рецепта, и в конце концов у вас получится пирог (интерфейс пользователя).

Данная книга посвящена главным образом SQL Server 2000, а не искусству создания интерфейса пользователя, поэтому здесь не будет подробного объяснения программы или процесса разработки. Я собираюсь показать вам, как превратить элементарный запрос к базе данных в Web-страницу. Что касается разработки интерфейса пользователя, то, возможно, я напишу об этом другую книгу.

Определение Web-страниц SQLSpyNet

Наш интерфейс будет относительно простым, всего 7–10 страниц, три из которых будут предназначены для установки и закрытия соединения с базой данных. Мы создадим также страницу приглашения и страницу запроса, чтобы пользователь смог ввести критерии поиска. Наш Web-узел будет содержать перечисленные ниже страницы.

- `Default.htm`. Это первая страница, открываемая при обращении к Web-узлу. Она будет некоторое время отображаться пользователям.
- `Login.asp`. Используется для подтверждения информации о регистрационном имени пользователя. Эту страницу можно увидеть.
- `Global.asa`. Начальная страница, содержащая информацию о соединении с базой данных в переменных, доступных во всем приложении. Это исключительно системный файл.
- `adovbs.inc`. Содержит константы, предоставленные Microsoft. Мы будем использовать сокращенную версию, полный файл нам не нужен. Это только системный включаемый (include) файл.

Термин *Включаемые файлы* — это файлы, содержащие функции, общие для всего узла. Такое название объясняется тем, что они подключаются к странице аналогично заголовочным файлам в C++ и Java.

- `Connection.asp`. Используется для фактического создания соединения с базой данных. Это только системный включаемый файл.
- `ConnectionClose.asp`. Используется для закрытия соединения с базой данных. Это только системный включаемый файл.
- `Welcome.asp`. Первая страница, отображаемая пользователям после успешной регистрации.
- `Search.asp`. С помощью этой страницы пользователи могут выполнять поиск в базе данных.

Такова базовая структура нашего Web-узла. Теперь коротко рассмотрим программное обеспечение, необходимое для решения поставленной задачи.

Конфигурирование компьютера для выполнения приложения SQLSpyNet

Прежде чем приступить к созданию реальной Web-страницы, необходимо подготовить компьютер к работе в качестве сервера. Для этого нужно установить некоторые программы.

Как отмечалось раньше, нужно установить PWS. Его можно найти в папке дополнительных программ инсталляционного компакт-диска Windows 98 или загрузить по адресу: `http://www.microsoft.com/windows/ie/pws/main.htm`.



В приложении А, "Установка Web-сервера", описано конфигурирование компьютера в качестве Web-сервера.

Вы, конечно, захотите использовать созданные мною файлы, чтобы не набирать код вручную. Необходимые файлы можно найти на Web-узле Издательского дома "Вильямс" (www.williamspublishing.com). Получите их и отложите в сторону, пока они не понадобятся.

Необходимо также установить ADO (ActiveX Data Objects). К счастью, это уже сделано при установке SQL Server 2000 как часть обновления MDAC. Обновление MDAC устанавливает также SQLOLEDB.

Единственное, что осталось, — установить Microsoft FrontPage. Он поставляется с Internet Explorer 5.0, поэтому, если Microsoft FrontPage еще не установлен, перезапустите установку Internet Explorer и выберите необходимый компонент из списка.

На заметку Вам придется использовать именно Microsoft FrontPage. Как отмечалось раньше, можно было бы использовать Блокнот, Word или любой текстовый редактор. Однако лучше используйте Microsoft FrontPage или же Visual InterDev.

И это все! Теперь вы готовы начать создание своего первого Web-узла. Приступим к делу. Но помните, если вы чего-либо не понимаете, вам придется самостоятельно исследовать неясный вопрос.

Установка Web-узла под управлением PWS

Перед созданием Web-узла нужно определить место хранения файлов.

1. Найдите на локальном компьютере поставляемый с PWS стандартный Web-узел. Его примерный вид — C:\Inetpub\wwwroot\.
2. В этой папке создайте новую папку SpyNet.
3. В папке SpyNet создайте две новые папки — Includes и Images. Полученная файловая система будет выглядеть примерно так, как показано на рис. 12.4.

В папке SpyNet будут храниться все основные файлы для Web-узла. В папке Includes будут находиться включаемые файлы, а в папке Images — изображения. При разработке (или копировании) каждой страницы я сообщу вам, куда их переносить, чтобы узел не оказался разрушенным. Сейчас поместите все изображения в папку Images.

Теперь у нас есть все, что нужно. Начинаем строить!

Создание новой учетной записи пользователя и установка соединения

Это совершенно новый Web-узел, поэтому пользователей, имеющих к нему доступ, еще нет.

Напоминаю: в ходе изучения материала главы 9, "Обеспечение безопасности базы данных Spy Net", была создана учетная запись пользователя SQLSpyNetUser. Сейчас мы создадим новую учетную запись, обладающую ограниченными правами доступа. Ведь мы не хотим, чтобы кто-нибудь взламывал наш узел, не так ли?



Мы создали учетную запись с ограниченными правами доступа, однако это еще не гарантирует полной безопасности. Чтобы найти приемлемый способ обеспечения безопасности узла, следует подробнее ознакомиться с вопросами безопасности в Web.

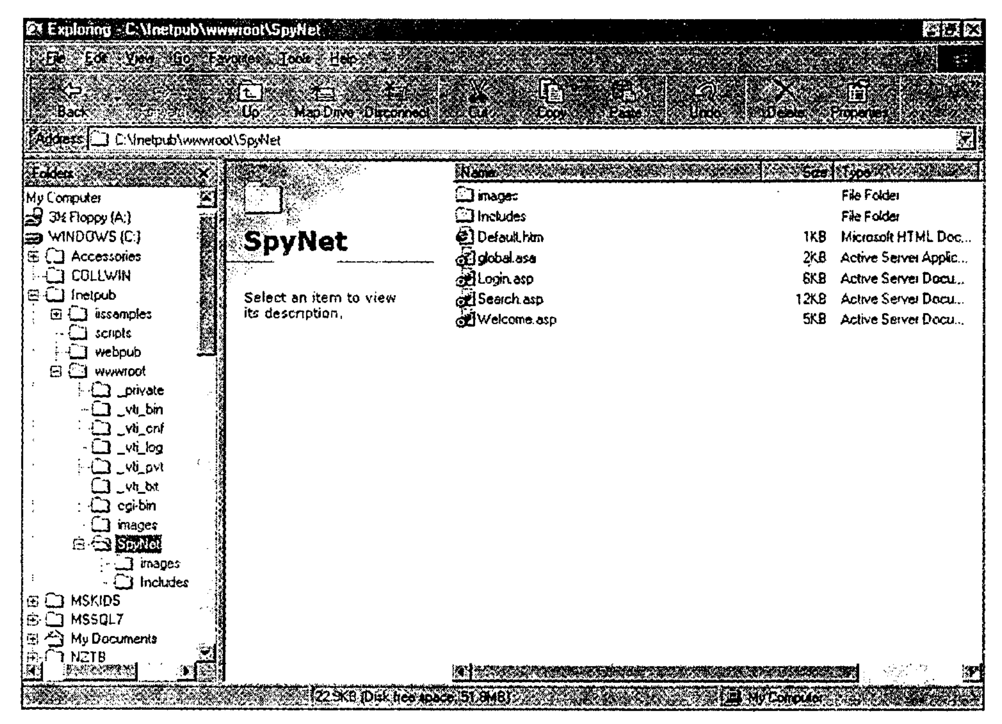


Рис. 12.4. Вид файловой системы перед началом создания Web-узла SQLSpyNet

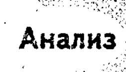
1. Создайте учетную запись пользователя с именем `SpyNetIntranetUser` и дайте ему легко запоминаемый пароль.
2. Новый пользователь должен иметь доступ к базе данных `SQLSpyNet`, однако он должен быть *только* членом роли `Public`.
3. Предоставьте новому пользователя единственное право — право доступа `SELECT` к таблице `Person` (причины этого станут ясны позже).

Создание нашей первой страницыGlobal.asa

Сначала создадим страницу `Global.asa`. В ASP-приложении это первая вызываемая системная страница, она содержит информацию, доступную для всего приложения. Поэтому назовите файл тем же именем (`Global.asa`) и введите в него текст листинга 12.1.

Листинг 12.1. Создание соединения с помощью файла Global.asa

```
<SCRIPT LANGUAGE="VBScript" RUNAT=="Server">
Sub Application_OnStart
    Application("Conn_ConnectionString")="Provider=SQLOLEDB.1";
    ↳Initial Catalog=SQLSpyNet;Data Source=GRUNTER/MYSQLSERVER;"
    Application("Conn_ConnectionTimeout")=15
    Application("Conn_CommandTimeout")=30
    Application("Conn_CursorLocation")=3
End Sub
</SCRIPT>
```



Чтобы программа выполнялась в вашей системе, нужно внести некоторые изменения. В строке 3 `Application("Conn_ConnectionString")` вместо `GRUNTER` необходимо ввести имя вашего сервера. После этого должно идти имя экземпляра (у меня — `MYSQLSERVER`).



Имена сервера и экземпляра можно посмотреть в *Enterprise Manager*. Открыв *Enterprise Manager*, вы увидите список серверов, выполняющихся в вашей системе (и сети, если есть соединение). Выберите один из них и скопируйте имя через буфер обмена.

Утилита *Enterprise Manager* рассматривается в главе 2, "Компоненты SQL Server 2000".

Обратите внимание на префиксы `Application` в некоторых строках (например, `Application("Conn_ConnectionString")`). Эти переменные с префиксами содержат информацию, доступную во *всем* приложении.

Страница `Global.asa` должна находиться в корневой папке Web-проекта.

Создание страницы `Default.htm`

Теперь нужно создать страницу `Default.htm`. По умолчанию в PWS это первая страница, отображаемая при обращении пользователя к узлу. `Default.htm` — это специальное имя, которое ищет PWS (или IIS). Можно конфигурировать систему на другое имя, однако `Default.htm` нас вполне устраивает.

На заметку

В странице `Default.htm` нет ничего особенного. Это обыкновенная страница HTML, без динамических компонентов. Но она ключевая в нашей разработке.

Создайте новую страницу и введите имя **Default.htm**. Затем введите в страницу текст листинга 12.2.

Листинг 12.2. Файл `Default.htm`

```
1:  <HTML>
2:      <HEAD>
3:          <TITLE>Spy Net Limited</TITLE>
4:          <META NAME="RedirectPage" HTTP-EQUIV="REFRESH"
            ➥CONTENT="0;URL=Login.asp">
5:      </HEAD>
6:      <BODY></BODY>
7:  </HTML>
```

При вызове эта страница перенаправляет нас (посмотрите на строку 4, особенно на дескриптор `CONTENT`) к странице `Login.asp`. Теперь для работы нашего Web-узла есть практически все.

Если обратиться к Web-узлу, введя в броузере адрес (`http://`*имя_сервера/имя_узла*, т.е. `http://grunter/SpyNet`), то `Default.htm` направит нас к странице `Login.asp`, которой пока еще не существует, и мы получим хорошо знакомую ошибку 404 (не найдена страница).

На заметку

Все необходимые файлы можно найти на Web-узле Издательского дома "Вильямс" (`www.williamspublishing.com`). Помните, что вам нужно изменить код файла `Global.asa`, чтобы Web-узел нормально функционировал в вашей системе. Для тех, кому больше нравится читать код с листа, я добавил его в книгу. Так что читайте и наслаждайтесь!

Файл `Default.htm` тоже расположен в корне Web-проекта. Это значит, что для поиска нового файла нам не нужно конфигурировать Web-сервер (PWS), файл будет найден по умолчанию.

Создание включаемых файлов

Сначала рассмотрим включаемые файлы и выясним, зачем они нам нужны. Использование этих файлов позволяет избежать многократного набора повторяющегося текста. Подпрограммы выполняют заданный набор операций; они создаются для того, чтобы их можно было при необходимости повторно вызывать. Что угодно, лишь бы не работать!

Существует несколько способов установки соединения с базой данных, а также несколько способов извлечения данных. По моему мнению, лучший из них — ADO.

Термин

С помощью *ADO* (ActiveX Data Objects) можно легко устанавливать соединение с базой данных, считывать и обновлять данные. Объектная модель ADO очень проста, она состоит главным образом из объектов `Connection`, `Command` и `Recordset`. Взаимодействуя, они предоставляют огромные возможности управления данными.

При разработке Web-узла ADO предоставляет больше гибкости — можно устанавливать соединение с базой данных, получать наборы данных и т.д.

Использование файла `Connection.asp` для организации ADO-подключения

Для создания с помощью ADO соединения с базой данных `SQLSpyNet` нужно предоставить объекту `Connection` (компонент ADO) правильные параметры, а именно: *имя сервера, имя базы данных, имя пользователя* и *пароль*. В листинге 12.3 приведен исходный текст этой страницы, названной достаточно просто — `Connection.asp`.

На заметку

Комментарии в VBScript начинаются с одинарной кавычки ('). Следовательно, текст после одинарной кавычки предназначен для разъяснения программы.

Листинг 12.3. Установка соединения с базой данных с помощью ADO

```
<%'Объявление переменной Conn
Dim Conn
'Проверка, существует ли соединение, если нет,
➥то ничего не делается
If Not isObject(Conn) Then
   'Если нет, то установить тип переменной как
   ➥ADO Connection Object
   Set Conn =Server.CreateObject("ADODB.Connection")
   With Conn
    'Задание параметров объекта Connection
    .ConnectionTimeout = Application("Conn_ConnectionTimeout")
    .CommandTimeout = Application("Conn_CommandTimeout")
    .CursorLocation = Application("Conn_CursorLocation")
   End With
End If%>
```

Анализ

Как видите, некоторым из этих параметров мы присвоили значения в файле Global.asa. Не присвоены значения только переменным username и password, потому что их должен ввести пользователь при регистрации. В файле Connection.asp мы передаем объекту Connection строковую информацию о соединении и таким образом создаем ADO-соединение с базой данных SQLSpyNet.

Разрыв соединения

Следующий включаемый файл, который мы рассмотрим, — страница ConnectionClose.asp (листинг 12.4). В ее функции входит очистка памяти после закрытия объекта Connection. Каждый разработчик компонентов промежуточного уровня скажет вам: "Всегда удаляйте объекты, с которыми закончили работать!"

Листинг 12.4. Файл ConnectionClose.asp

```
1: <%If isObject(Conn) Then
2:     Conn.Close
3:     Set Conn = Nothing
4: End If%>
```

Анализ

При включении ConnectionClose.asp в нашу страницу мы сначала закрываем соединение (строка 2), а затем удаляем объект из памяти (строка 3).

Обеспечение правильной работы VBScript с помощью файла adovbs.inc

Файл adovbs.inc — третий, и последний, включаемый файл. В языке VBScript нет *сильной типизации*, к тому же VBScript не предоставляет доступ ко всем библиотекам Visual Basic, поэтому настоятельно рекомендую держать все константы (переменные, не изменяющие свое значение) в одном месте.

Термин

Понятие *сильной типизации* относится к используемым в языке типам данных. Язык VBScript не является сильно типизированным, потому что в нем все переменные, пока им не присвоено первое значение, интерпретируются как Variant (специальный тип, содержащий что угодно). С другой стороны, Visual Basic 6.0 — сильно типизированный язык, потому что в нем тип задается при объявлении переменной, т.е. в момент создания.

Мы используем константы, потому что их значениями могут быть слова естественного языка, а не номера, удобные для интерпретатора, но неудобные для человека.

В нашем узле мы используем сокращенную версию файла adovbs.inc (листинг 12.5). Открытие файла при каждой загрузке страницы занимает довольно много времени, поэтому немного сократим его, чтобы увеличить быстродействие.

Листинг 12.5. Константы VBScript в файле adovbs.inc

```
<%
'------------------------------------------------------------------
' ((SOME)Microsoft ADO
'
' Включаемый файл с константами ADO для VBScript

'------------------------------------------------------------------
'---- Значения CursorTypeEnum ----
Const adOpenStatic = 3

'---- Значения LockTypeEnum ----
Const adLockReadOnly = 1

'---- Значения CursorLocationEnum ----
Const adUseClient = 3

'---- Значения ObjectStateEnum ----
Const adStateClosed = &H00000000
Const adStateOpen = &H00000001

'---- Значения ParameterDirectionEnum ----
Const adParamInput = &H0001
Const adParamOutput = &H0002

'---- Значения CommandTypeEnum ----
Const adCmdStoredProc = &H0004

'---- Значения DataTypeEnum ----
Const adVarChar = 200
%>
```

Подтверждение регистрации пользователя

Эта страница доставит вам наибольшее наслаждение! Она принимает имя и пароль пользователя и передает их объекту ADO Connection. Код страницы Login.asp приведен в листинге 12.6.

Листинг 12.6. Проверка прав доступа пользователя с помощью страницы Login.asp

```
<%@Language="VBScript "%>
<%'Option Explicit означает объявление всех переменных
Option Explicit

'Response.Expires означает, что страница
 ➥не будет кэширована в броузере
 Response.Expires = -1000

 'Response.Buffer означает, что страница буферизована.


 Response.Buffer =True

 'Эта переменная доступна на уровне страницы, поэтому мы можем
 'возвращать свое сообщение об ошибке.
```

```
'Смотрите ниже
Dim sErrorMessage

'Пользователь щелкнул на кнопке?
'Если да, то вызывается функция dologin
If Len(Trim(Request.Form("M")))<>0 Then
   'Если регистрационная информация верна,
   ➥то пользователь направляется к странице Welcome
   If DoLogin Then
      Response.Redirect("Welcome.asp")
   End If
End If%>

<HTML>
   <HEAD>
      <META NAME="GENERATOR" Content="Microsoft
                  ➥Visual Studio 6.0">
      <TITLE>Spy Net Limited</TITLE>
   </HEAD>

   <SCRIPT LANGUAGE="JavaScript">
      function CheckLogin(){
         /*Этот сценарий только проверяет, ввел ли пользователь
         свое имя перед получением страницы.
         Оно вызывается кнопкой Login в нижней части страницы.
         Если пользователь ввел имя, отображаем страницу */
         if (formMain.UserName.value == "")
            {
            alert('Oops, you have forgotten to type your
                 ➥User Name. Please enter to continue.');
            formMain.UserName.focus();
            }
         else
            formMain.submit();
      }
   </SCRIPT>

   <BODY LEFTMARGIN="0" TOPMARGIN="0"
   ➥MARGINWIDTH="0" MARGINHEIGHT="0"
   ➥BGCOLOR="#666666" LANGUAGE="JavaScript"
   ➥ONLOAD="formMain.UserName.focus();">
      <TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
      ➥WIDTH="100%" BORDERCOLOR="RED">
         <TR>
            <TD WIDTH="411"><IMG SRC="images/
            ➥topstrip.jpg" WIDTH="411"
            ➥HEIGHT="147" VSPACE="0" HSPACE="0"></TD>
            <TD WIDTH="38"><IMG SRC="images/
            ➥topstripmid.jpg" WIDTH="38"
            ➥HEIGHT="147" VSPACE="0" HSPACE="0"></TD>
            <TD WIDTH><IMG SRC="images/
            ➥topstripright.jpg" WIDTH="100%"
            ➥HEIGHT="147" VSPACE="0" HSPACE="0"></TD>
         </TR>
      </TABLE>

```

```
<TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
↳BORDERCOLOR="GREEN">
   <TR>
      <TD ROWSPAN="3"> <IMG SRC="images/figure.gif"
      ↳VSPACE="20" HSPACE="0" WIDTH="172"
      ↳HEIGHT="285"></TD>
      <%If Len(Trim(sErrorMessage))<>0 Then
         Response.Write("<TD WIDTH=""324"" HEIGHT=""99""
         ↳VALIGN=""CENTER""><FONT FACE=""arial,
         ↳verdana,helvetica"" SIZE=""2""
         ↳COLOR=""#ffffff""> " & sErrorMessage &
         ↳"</FONT></TD>")
      Else
         Response.Write("<TD HEIGHT=""99""> </TD>")
      End If%>
      <TD ROWSPAN="3"><IMG SRC="images/logo.jpg "
      ↳WIDTH="295" HEIGHT="290"></TD>
   </TR>
   <TR>
      <TD valign="top"><IMG SRC="images/title.jpg"
      ↳WIDTH="324" HEIGHT="63"></TD>
   </TR>
   <TR>
      <TD> </TD>
   </TR>
</TABLE>

<FORM NAME="formMain" ACTION="Login.asp" METHOD="POST">
   <TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
   ↳WIDTH="100%"
   ↳BORDERCOLOR="BLUE">
      <TR>
         <TD ROWSPAN="5" valign="top">
            <IMG SRC="images/bottomstrip.jpg"
            ↳WIDTH="100%" HEIGHT="162"></TD>
         <TD WIDTH="235">
            <IMG SRC="images/logintopstrip.jpg"
            ↳WIDTH="235" HEIGHT="17"></TD>
         <TD ROWSPAN="5" valign="top">
            <IMG SRC="images/bottomstrip.jpg"
            ↳WIDTH="100%" HEIGHT="162"></TD>
      </TR>
      <TR>
         <TD BGCOLOR="#000000" HEIGHT="17">
            <FONT COLOR="#ff8c00" STYLE=
            ↳"FONT-FAMILY: Verdana;
            ↳FONT-SIZE: x-small;">Username:  
            ↳       
            ↳ Password:</FONT>
         </TD>
      </TR>
      <TR>
         <TD BGCOLOR="#000000" HEIGHT="17">
            <INPUT TYPE="TEXT" NAME="UserName" SIZE="15">
            ↳ <INPUT TYPE="PASSWORD" NAME="UserPwd"
```

```
                    ↳SIZE="15">
                 </TD>
              </TR>
              <TR>
              <TD BGCOLOR="#000000" HEIGHT="17" ALIGN="RIGHT">
              <IMG LANGUAGE="JavaScript" ONCLICK="CheckLogin();"
              ↳STYLE="Cursor:Hand" SRC="Images/Login.gif"
              ↳VALUE="Login" ID="Login"
              ↳NAME="Login" WIDTH="101" HEIGHT="13"> 
            </TD>
         </TR>
         <TR>
             <TD>
                 <IMG SRC="images/loginbottomstrip.jpg"
                 ↳HEIGHT="83" WIDTH="100%">
              </TD>
           </TR>
         </TABLE>
         <INPUT TYPE="HIDDEN" NAME="M" SIZE="15" VALUE="Login">
       </FORM>
     </BODY>
   </HTML>

   <%Function DoLogin
   'Этот оператор позволяет перехватывать ошибки
   On Error Resume Next
      'bResult - эта переменная принимает значения True или False
      Dim bResult
      'Переменная sConn используется для создания
      ↳строковой информации о соединении, которая нужна для
      'установки соединения с базой данных
      Dim sConn
      'Это нужные включаемые файлы!!%>
      <!--#INCLUDE FILE = "Includes/adovbs.inc" -->
      <!--#INCLUDE FILE = "Includes/Connection.asp" -->
      <%If isObject(Conn)Then
         sConn = Application("Conn_ConnectionString")
         ↳ & ";User Id=" &
         ↳Request.Form("UserName") & ";PASSWORD=" &
         ↳Request.Form("UserPwd") & ";"
         Conn.Open sConn
         If Conn.State =adStateOpen Then
            bResult =True
         Else
            bResult =False
         End If
      End If

      If Err.number<>0 Then
         'Это код ошибки при неудачной регистрации
         If Err.number =-2147217843 Then
            bResult =False
            sErrorMessage ="Oops, we cannot log you in.
            ↳Please check the credentials you have supplied."
         Else
            bResult =False
```

```
            sErrorMessage ="Oops, something has gone
            ➥wrong internally. Please try again, and if
            ➥the problem persists contact
            ➥your System Administrator."
         End If
      End If

      'Если попытка успешная, информация сохраняется
      'в переменной сеанса, поэтому ею могут воспользоваться
      'другие страницы соединения с базой данных
      If bResult Then
         Session("UserName") = Request.Form("UserName")
         Session("UserPwd") = Request.Form("UserPwd")
      End If
'************************************************************
      'ВНИМАНИЕ:
      'ХРАНИТЬ ПАРОЛЬ В ПЕРЕМЕННОЙ СЕАНСА НЕ РЕКОМЕНДУЕТСЯ
      'Переменные сеанса предназначены только для зарегистрированно-
го
      'пользователя, они не зашифрованы. Поэтому их легко взломать.
      'Для реального узла подберите другую архитектуру
'**************************************************************
***

      'Возврат результатов попытки регистрации
      DoLogin =bResult
End Function%>
```



В этом листинге наиболее важно, как ADO устанавливает соединение с базой данных. Если попытка успешна (имя пользователя и пароль верны), пользователь допускается и направляется к странице `Welcome.asp`. Если неуспешна — получает сообщение об ошибке, информирующее о неуспехе попытки.

Два текстовых поля ввода на экране принимают информацию от пользователя и, когда она передается обратно на сервер (когда пользователь щелкает на кнопке **Login**), программа принимает ее и пытается установить соединение.

Где все это происходит? Посмотрите в функцию `DoLogin`, вы увидите, что она принимает имя пользователя и пароль и передает их странице `Connection.asp`. Если соединение устанавливается успешно, то пользователь допускается, а если нет, он должен сделать еще одну попытку.

Файл `Login.asp` расположен в корне Web-проекта.

Страница `Welcome.asp` приветствует пользователя

С этой страницей будут работать пользователи, т.е. перемещаться по узлу, выполнять поиск, заканчивать сеанс работы и, возможно, в будущем обновлять данные, хранящиеся в базе данных.

Страница `Welcome.asp` (листинг 12.7) предоставляет пользователю разнообразную информацию. Но с точки зрения разработки она довольно проста. Понять ее нетрудно, так как представленный в ней текст — практически “чистый” HTML.

Листинг 12.7. Страница Welcome.asp

```
<%@Language="VBScript "%>
<%'Option Explicit означает объявление всех переменных
Option Explicit

'Response.Expires означает, что страница
 ↳не будет кэширована в броузере
Response.Expires =-1000

 'Response.Buffer означает, что страница буферизована.


 Response.Buffer =True

 'Если пользователь хочет закончить работу,
 ↳переменная сеанса разрушается
 'При переходе к следующей строке
 ↳управление передается Login.asp
 'Как просто, не правда ли?
 If Len(Trim(Request("Destroy")))<>0 Then
    If CInt(Trim(Request("Destroy")))=1 Then
       Session("UserName")=""
    End If
 End If

 'Проверка, есть ли в переменной сеанса имя пользователя.
 'Если нет, то пользователь направляется к странице Login.asp.
 ↳Пользователь будет вынужден зарегистрироваться еще раз.
 'Если бы не было этой процедуры, то любой смог бы просто набрать
 'в окне броузера URL страницы и попасть на узел
 If Len(Trim(Session("UserName ")))=0 Then
    'Почему мы только проверяем имя пользователя?
    ↳Потому что пользователь может не иметь пароля!
    Response.Redirect("Login.asp")
 End If%>

 <HTML>
    <HEAD>
       <META NAME="GENERATOR" Content="Microsoft
        ↳Visual Studio 6.0">
       <TITLE>Spy Net Limited</TITLE>
    </HEAD>

    <SCRIPT LANGUAGE="JavaScript">
       function VerifyLogout(){
          var bConfirm = window.confirm('This
          ↳will log you out of Spy Net.
          ↳Are you sure you wish to continue?');
          if (!bConfirm)
             window.event.returnValue = false
       }
    </SCRIPT>

    <STYLE TYPE="text/css">
       A:link       {color: #ffffff;text-decoration:none}
       A:visited    {color: #ffffff;text-decoration:none}
       A.b          {color: #ffffff;text-decoration:none;}
       A.r          {color: #ffffff;text-decoration:none;}
```

```
        A.bb         {color: #ffffff;text-decoration:none;}
        A.bb:visited {color: #ffffff;text-decoration:none;}
        A:hover      {color: #ff8c00;text-decoration:underline;}
        A.bb:hover   {color: #ff8c00;}
    </STYLE>

    <BODY LEFTMARGIN="0" TOPMARGIN="0"
    ➥MARGINWIDTH="0" MARGINHEIGHT="0"
    ➥BGCOLOR="#666666">

        <TABLE CELLPADDING="0" CELLSPACING="0"
        ➥BORDER="0" WIDTH="100%">
            <TR>
                <TD VALIGN="top"><IMG SRC="images/
                ➥welcometopstrip.jpg" WIDTH="100%"
                ➥HEIGHT="101" VSPACE="0" HSPACE=="0"></TD>
                <TD WIDTH="38" VALIGN="top">
                    <IMG SRC="images/welcometopstripmid.jpg"
                    ➥WIDTH="38" HEIGHT="101" VSPACE="0"
                    ➥HSPACE="0"></TD>
                <TD WIDTH="303" VALIGN="top"><IMG SRC="images/
                ➥menu.gif" WIDTH="303" HEIGHT="100"
                ➥useMAP="#menu" BORDER=="0" >>
                    <MAP NAME="menu">
                        <AREA SHAPE="rect" COORDS="156,43,210,56"
                        ➥HREF="search.asp" TITLE="Search"
                        ➥ALT="Search">
                        <AREA SHAPE="rect" COORDS="97,44,151,55"
                        ➥ONCLICK="VerifyLogout();"
                        ➥HREF="Welcome.asp?Destroy=1"
                        ➥TITLE="Logout" ALT="Logout">
                        <AREA SHAPE="rect" COORDS="38,44,92,56"
                        ➥HREF="welcome.asp" TITLE="Home" ALT="Home">
                    </MAP>
                </TD>
            </TR>
        </TABLE>

        <TABLE CELLPADDING="0" CELLSPACING="0"
        ➥BORDER="0" WIDTH="600">
            <TR>
                <TD ROWSPAN="2" VALIGN="top"> 
                    <IMG SRC="images/figure.gif"
                    ➥VSPACE="20" HSPACE="0"
                    ➥WIDTH="172" HEIGHT="285">
                </TD>
                <TD HEIGHT="99" VALIGN="top" COLSPAN="2">
                    <IMG SRC="images/welcome.gif"
                    ➥WIDTH="167" HEIGHT="77">
                </TD>
            </TR>
            <TR>
                <TD WIDTH="30"> </TD>
                <TD VALIGN="top">
                    <FONT FACE="arial, verdana, helvetica"
                    ➥SIZE="2" COLOR="#ffffff">
                        <P>
                            Welcome to Spy Net LTD. This site
```

```
                              ↳runs in conjunction with the
                              ↳SQL Server 2000 Database Development
                              ↳from Scratch book by
                              ↳<A HREF="mailto:rob-marg@xtra.co.nz">
                              ↳Rob Hawthorne</A>.<BR><BR>
                              The development of this site is part of the
                              ↳book, and is designed to give you an
                              ↳overview of how easy it is to create
                              ↳a SQL Server 2000 database and then
                              ↳implement that on the web!
                           </P>
                           <P>
                              On the top right of the screen,
                              ↳you will see three buttons
                              ↳"Home", "Logout", and "Search".
                              <LI>The "Home" button will always bring
                              ↳you back here.
                              <LI>The "Logout" button will log you
                              ↳out of the system, and return you
                              ↳to the login page.
                              <LI>The "Search" button will take you to
                              ↳the search screen, so you can perform a
                              ↳search for People in the system.
                           </P>
                           <P>
                              So have a play and familiarize
                              ↳yourself with the site.<BR>
                              If you wish to add to it later, that
                              ↳would be great!<BR>
                              Some suggested ideas are
                              <LI>Have a data entry screen for
                              ↳Spies and Bad Guys
                              <LI>Have a reports screen to see who
                              ↳is on what activity
                              <LI>Implement a notification system to a Spy
                              ↳when they are reassigned i.e. an Email
                            </P>
                            <P>
                               PS: If your PC supports it, this site looks
                               ↳great in 16-bit (or higher color)
                               ↳and a 1024x768 (or greater)resolution.
                            </P>
                            </FONT>
                            <FONT FACE="arial, verdana, helvetica"
                            ↳SIZE="2" COLOR="#ff8c00">
                            <P>
                               This site's awesome graphics were provided
                               ↳by a very talented Graphic Artist.
                               ↳Thank you Jillian for all your hard work!
                            </P>
                         </FONT>
                      </TD>
                   </TR>
                   <TR>
                      <TD> </TD>
                   </TR>
                </TABLE>
```

```
        <TABLE CELLPADDING="0" CELLSPACING="0"
        ↳BORDER="0" WIDTH="100%">
           <TR>
              <TD ROWSPAN="3" VALIGN="top">
                 <IMG SRC="images/bottomsTRip.jpg"
                 ↳WIDTH="100%" HEIGHT="162">
              </TD>
           </TR>
        </TABLE>
     </BODY>
</HTML>
```

Страница `Welcome.asp` также находится в корне Web-проекта.

Создание страницы `Search.asp` для считывания данных из базы данных SQLSpyNet

Эта страница реализует функциональные возможности нашего Web-узла. Она позволяет динамически искать информацию в базе данных, а в будущем — создавать гиперссылки для просмотра и редактирования данных.

Что нужно для запуска страницы `Search.asp`? В первую очередь — выяснить, что мы ищем? Например, человека, разведчика или адрес?

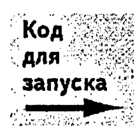
К счастью, я уже решил этот вопрос. Сейчас мы реализуем поиск в таблице `Person`, что позволит найти человека по имени или фамилии. Даже более того: пользователь сможет ввести только первые буквы имени (или фамилии) и получить ответ. Например, если пользователь введет `Ha`, то получит ответ `Hawthorne` или `Harrison`.

Создание хранимой процедуры для поиска в SQLSpyNet

Создадим хранимую процедуру. Вы можете делать это с помощью средств как Query Analyzer, так и Enterprise Manager. Я уверен, вы уже отлично знаете оба эти инструмента!

Введите код хранимой процедуры, приведенный в листинге 12.8.

Листинг 12.8. Хранимая процедура для поиска в таблице `Person`




```
CREATE PROCEDURE PersonSearch
   @FirstName VARCHAR(50) = NULL,
   @Surname VARCHAR(50) = NULL
AS
   SELECT
      PersonID,
      Firstname +' ' + Surname AS Fullname,
      dbo.DateFormatter(DOB,' ') AS DOB,
      PhoneNo
   FROM Person
   WHERE (@FirstName IS NULL OR
      Firstname LIKE @FirstName +'%')
      AND (@Surname IS NULL OR Surname LIKE @Surname +'%')
   ORDER BY Firstname, Surname
GO
```

Анализ

В этой процедуре незнакомы только строки 11 и 12. Выражение `@FirstName IS NULL` позволяет определить, имеет ли переменная значение. Если ее значение равно `NULL`, то фраза `OR` не выполняется. То же справедливо и для `@Surname`. Поэтому процедуру можно выполнить, даже не передав ей ни одного параметра, в этом случае она возвратит из таблицы `Person` всех! Однако если передать параметрам значения, то поиск будет ограничен этими значениями. Параметр `'%'` в конце оператора `LIKE` при поиске означает присоединение к введенному параметру произвольной строки символов.

Созданную процедуру нужно проверить (не сомневаюсь, что вы и сами подумали об этом). Выполните код листинга 12.9 и убедитесь, что запросы возвращают то, что ожидается.

Листинг 12.9. Проверка поиска

Код для запуска

```
EXEC PersonSearch @Surname = 'T'
GO
EXEC PersonSearch
GO
```

Анализ

Первый поиск ограничен записями о людях с фамилиями, начинающимися с буквы `T`. Второй запрос возвращает все, что есть в таблице, поскольку процедуре не переданы параметры, ограничивающие поиск.

Предоставление пользователю права на поиск

Теперь, когда хранимая процедура создана, мы должны предоставить нашему новому пользователю право доступа к ней. Выполните запрос, приведенный в листинге 12.10, или запустите Enterprise Manager и присвойте пользователю право выполнения процедуры.

Листинг 12.10. Предоставление пользователю права на поиск

Код для запуска

```
GRANT EXECUTE ON PersonSearch TO SpyNetIntranetUser
```

Итак, процедура создана, пользователь имеет право выполнять ее, теперь нужно создать форму для фактического поиска данных.

Создание формы поиска

Страница `Search.asp` состоит из двух частей. Первая — базовый текст HTML, включая поля ввода текста пользователем для задания условий поиска. Вторая — функция, осуществляющая поиск. Она устанавливает соединение с базой данных и выполняет поиск.

Внутри этой функции вводятся два новых объекта ADO, которые вы увидите в листинге 12.11.

- Command. Позволяет указать, что мы создаем хранимую процедуру и используем для поиска определенные параметры. С помощью объекта Command создается команда, которая выполняется в базе данных.
- Recordset. Позволяет перенести набор записей обратно. Получив набор записей, их можно отобразить на экране.

Объекты ADO обладают многими свойствами, которыми можно манипулировать. Об этом написана не одна книга! Чтобы вам было легче продвигаться дальше, я добавил в текст страницы несколько комментариев, отмеченных одинарными кавычками (') и звездочками (*). Они покажут вам, где происходит действие.

Листинг 12.11. Страница Search.asp

```
<%@Language="VBScript "%>
<%'Option Explicit означает объявление всех переменных
Option Explicit

'Response.Expires означает, что страница
➥не будет кэширована в броузере
Response.Expires =-1000

 'Response.Buffer означает, что страница буферизована.

Response.Buffer =True

'Если пользователь хочет закончить работу,
➥переменная сеанса разрушается
'При переходе к следующей строке
➥управление передается Login.asp
'Как просто, не правда ли?
If Len(Trim(Request("Destroy")))<>0 Then
   If CInt(Trim(Request("Destroy"))) = 1 Then
      Session("UserName") = ""
   End If
End If

'Проверка, есть ли в переменной сеанса имя пользователя.
'Если нет, то пользователь направляется на Login.asp.
➥Пользователь будет вынужден зарегистрироваться еще раз.
'Если бы не было этой процедуры, то любой смог бы просто ввести
➥в окне броузера адрес страницы и попасть на узел
If Len(Trim(Session("UserName ")))=0 Then
   'Почему мы проверяем только имя пользователя?
   ➥Потому что пользователь может не иметь пароля!
   Response.Redirect("Login.asp")
End If

'Эта переменная доступна на уровне страницы,
➥она будет использоваться в разных местах
Dim sErrorMessage%>
<HTML>
   <HEAD>
      <META NAME="GENERATOR" Content="Microsoft
      ➥Visual Studio 6.0">
      <TITLE>Spy Net Limited</TITLE>
   </HEAD>

```

```
    <SCRIPT LANGUAGE="JavaScript">
       function VerifyLogout(){
          var bConfirm = window.confirm('This will
          ↳log you out of Spy Net.
          ↳Are you sure you wish to continue?');
          if (!bConfirm)
             window.event.returnValue = false
       }
    </SCRIPT>

    <BODY LEFTMARGIN="0" TOPMARGIN="0"
    ↳MARGINWIDTH="0" MARGINHEIGHT="0"
    ↳ONLOAD="formMain.Firstname.focus();" BGCOLOR="#666666">
       <FORM NAME="formMain" ACTION="Search.asp" METHOD="POST">
          <INPUT TYPE="HIDDEN" NAME="M" VALUE="DoSearch">
          <TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
          ↳BORDERCOLOR="RED" WIDTH="100%">
             <TR>
                <TD VALIGN="top">
<IMG SRC="images/welcometopsTRip.jpg"
↳WIDTH="100%" HEIGHT="101"
↳VSPACE="0" HSPACE="0"></TD>
                        <TD WIDTH="38" VALIGN="top">
<IMG SRC="images/welcometopsTRipmid.jpg"
↳WIDTH="38" HEIGHT="101"
↳VSPACE="0" HSPACE="0"></TD>
                        <TD WIDTH="303" VALIGN="top">
                   ↳<IMG SRC="images/menu.gif"
                   ↳WIDTH="303" HEIGHT="100" USEMAP="#menu"
                   ↳BORDER="0">
                   <MAP NAME="menu">
                      <AREA SHAPE="rect" COORDS="156,43,210,56"
                      ↳HREF="search.asp" TITLE="Home"
                      ↳ALT="Home">
                      <AREA SHAPE="rect" COORDS="97,44,151,55"
                      ↳ONCLICK="VerifyLogout();"
                      ↳HREF="Search.asp?Destroy=1"
                      ↳TITLE="Logout" ALT="Logout">
                      <AREA SHAPE="rect" COORDS="38,44,92,56"
                      ↳HREF="welcome.asp" TITLE="Home"
                      ↳ALT="Home">
                   </MAP>
                </TD>
             </TR>
          </TABLE>

          <TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
          ↳WIDTH="600" BORDERCOLOR="GREEN">
             <TR>
                <TD ROWSPAN="2"> <IMG SRC="images/figure.gif"
                ↳VSPACE="20" HSPACE="0" WIDTH="172"
                ↳HEIGHT="285"></TD>
                <TD VALIGN="top" COLSPAN="2" HEIGHT="77">
                   <IMG SRC="images/search.gif" WIDTH="167"
                   ↳HEIGHT="77"></TD>
             </TR>
             <TR>
```

```
                    <TD WIDTH="30"> </TD>
                    <TD VALIGN="top" HEIGHT="200">
                       <FONT COLOR="#ff8c00"
                       ➥STYLE="FONT-FAMILY: Verdana;
                       ➥FONT-SIZE: x-small;">Firstname: 
                       ➥      
                       ➥      
                       ➥      
                       ➥      
                       ➥Surname:</FONT><BR>
                       <INPUT TYPE="TEXT" NAME="Firstname" SIZE="28"
                       ➥ALIGN="right"
                       ➥VALUE="<%=Request.Form("FirstName")%>">
                       ➥  <INPUT TYPE="TEXT"
                       ➥NAME="Surname" SIZE="28" ALIGN="right"
                       ➥VALUE="<%=Request.Form("Surname")%>">
                       <BR><BR>
                       <INPUT TYPE="Image" SRC="Images/go.gif"
                       ➥VALUE="Search" ID="Search" NAME="Search"
                       ➥ALIGN="right" WIDTH="54" HEIGHT="10">
                       <BR>
                       <%'Если страница была представлена, значит,
                       ➥скрытое значение M существует, поиск тоже
                       ➥(см. верхнюю часть HTML
                       'под открытием дескриптора формы)
                       If Len(Trim(Request.Form("M")))<>0 And
                       ➥UCase(Trim(Request.Form("M")))="DOSEARCH"
                       ➥Then%>
                          <!--#INCLUDE FILE = "Includes/
                          ➥Connection.asp" -->
                          <!--#INCLUDE FILE = "Includes/
                          ➥adovbs.inc" -->
                          <%'Файл Connection.asp включен здесь для
                          ➥того, чтобы обе функции (см. ниже)
                          'могли видеть объект Conn
                          Call DoPersonSearch
                          'Теперь закрываем объект Connection
                          ➥(Всегда очищайте память!)%>
                       <!--#INCLUDE FILE =
                       ➥"Includes/ConnectionClose.asp" -->
                       <%End If%>
                    </TD>
                 </TR>
              </TABLE>

              <TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0"
              ➥BORDERCOLOR="BLUE" WIDTH="100%">
                 <TR>
                    <TD ROWSPAN="3" VALIGN="top">1
                       <IMG SRC="images/bottomsTRip.jpg" WIDTH="100%"
                       ➥HEIGHT="162"></TD>
                 </TR>
              </TABLE>
           </FORM>
        </BODY>
     </HTML>
```

```
<%'*************************************************************
'Здесь устанавливается соединение с базой данных
'*************************************************************
Function DoLogin
'Этот оператор позволяет перехватывать ошибки
On Error Resume Next
   'bResult - эта переменная принимает значение
   ↳True или False и используется в функции
   Dim bResult
   'Переменная sConn используется для определения
   ↳строковой информации
   ↳о соединении, необходимой для
   'создания соединения с базой данных
   Dim sConn

   If isObject(Conn)Then
      sConn = Application("Conn_ConnectionString")
      ↳ & ";User Id=" &
      ↳Session("UserName") & ";PASSWORD=" &
      ↳Session("UserPwd") & ";"
      Conn.Open sConn
      If Conn.State = adStateOpen Then
         DoLogin = True
      Else
         DoLogin = False
      End If
   End If

   If Err.number<>0 Then
      'Это код ошибки при неудачной регистрации
         If Err.number = -2147217843 Then
            DoLogin = False
            sErrorMessage = "Oops, we cannot log you in.
            ↳Please check the credentials you have supplied."
         Else
            DoLogin = False
            sErrorMessage ="Oops, something has gone
            ↳wrong internally.
            ↳Please try again, and if the problem persists
            ↳contact your System Administrator."
        End If
    End If
End Function

'************************************************************
'Здесь осуществляется поиск
'************************************************************
Sub DoPersonSearch
On Error Resume Next
   'Используется для подтверждения регистрационной информации
   ↳пользователя с помощью вызова функции DoLogin (см. ниже)
   Dim bValid
   'Используется для проверки, не произошла ли ошибка.
   ↳Если да, то bError примет значение True (см. ниже)
   Dim bError
   'Переменная Cmd предназначена для объекта ADO Command
   Dim Cmd
   'Переменная rs предназначена для объекта ADO Recordset
```

```
Dim rs
'Эта переменная содержит стиль шрифта,
↳поскольку я слишком ленив, чтобы записывать
↳его снова и снова
Dim sFontStyle
'Эти две переменные содержат номер текущей страницы и
↳общее количество страниц
Dim iPageCurrent, iPageCount

'Переменная ошибки устанавливается в False
↳на случай, если ее предыдущее значение было равно True
bError = False

'Переменная стиля шрифта
sFontStyle = "<FONT FACE=""arial, verdana,
↳helvetica"" SIZE=""2""
↳COLOR=""#ffffff"">"

'Переменная bValidUser получит из функции DoLogin
↳значение true или false
bValid = DoLogin

'Если пользователь "неправильный", пошлем ему сообщение
↳и выйдем отсюда
If Not bValid Then
   Response.Write(sFontStyle & sErrorMessage & "</FONT>")
   Exit Sub
End If

'Если до сих пор все было в порядке, выполняем поиск
Set Cmd = Server.CreateObject("ADODB.Command")
Set rs = Server.CreateObject("ADODB.RecordSet")
rs.CursorLocation = adUseClient

With Cmd
   .ActiveConnection = Conn
   .CommandType = adCmdStoredProc
   .CommandText = "dbo.PersonSearch"
   'Выполняем процедуру с и получаем результат
   If Len(Trim(Request.Form("FirstName ")))<>0 Then
      'Если пользователь ввел значение, выполняем
      ↳процедуру с этим значением
      .Parameters.Append
.CreateParameter("",adVarChar,adParamInput,50,Trim(
↳Request.Form("FirstName")))
   Else
      'Если значение не введено,
      ↳тоже выполняем процедуру, однако передаем Null
      .Parameters.Append
      ↳.CreateParameter("",adVarChar,adParamInput,50,Null)
      End If
      'То же самое для Surname
      If Len(Trim(Request.Form("Surname")))<>0 Then
         .Parameters.Append
         ↳.CreateParameter("",adVarChar,adParamInput,50,
         ↳Trim(Request.Form("Surname")))
      Else
         .Parameters.Append
```

```
                ➥.CreateParameter("",adVarChar,
                ➥adParamInput,50,Null)
            End If

'*****************************************************************
    'ПРЕДУПРЕЖДЕНИЕ: Мы не используем команду parameters.refresh
    ➥и не именуем параметры явно,
    'поэтому мы должны направлять параметры в процедуру
    ➥ТОЧНО в той же последовательности, как они были объявлены.
    'Почему бы не воспользоваться командой parameters.refresh?
    ➥Потому что тогда придется еще раз обращаться к SQL Server
    ➥для получения всей информации о параметрах для объекта
    ➥Command, а чем меньше обращений к серверу, тем лучше.
'*****************************************************************

        'Присвоение набору записей значений,
        ➥возвращенных объектом Command
        Set rs = .Execute
    End With

    If Err.number <>0 Then
        'Это код ошибки при отсутствии права доступа Execute
        If Err.number = -2147217911 Then
            sErrorMessage = "Oops, you do not have permission to
            ➥perform this functionality.
            ➥See your System Administrator for assistance."
            bError = True
        Else
            sErrorMessage ="Oops, something has gone
            ➥wrong internally.
            ➥Please try again, and if the problem
            ➥persists contact your System Administrator."
            bError = True
        End If
    End If

    'Вызывается только при ошибке.
    ➥Ошибка может состоять в отсутствии
    ➥права Execute для процедуры.
    'Это может быть также другая ошибка.
    If bError Then
        Response.Write(sFontStyle & sErrorMessage & "</FONT>")
        'Не забудьте все очистить
        rs.Close
        Set rs = Nothing
        Set Cmd = Nothing
        'Все слишком хорошо, поэтому давайте выходить отсюда
        Exit Sub
    End If

    rs.PageSize = 3

    'Проверка, не пуста ли переменная rs
    'Если да, то сообщаем пользователю,
    ➥очищаем объекты и выходим
    If rs.EOF Then
        sErrorMessage = "Your search produced no results.
        ➥Please re-enter some new search criteria, and try again."
```

```
      Response.Write(sFontStyle & sErrorMessage & "</FONT>")
      'Очищаем все
      rs.Close
      Set rs = Nothing
      Set Cmd = Nothing
      Exit Sub
   End If

   iPageCurrent = Request.Form("PageCurrent")
   If Len(Trim(Request.Form("Next.x")))<>0 Then
      iPageCurrent =iPageCurrent + 1
   ElseIf Len(Trim(Request.Form("Previous.x")))<>0 Then
      iPageCurrent = iPageCurrent -1
   Else
      iPageCurrent = 1
   End If

   rs.AbsolutePage = iPageCurrent
   iPageCount = rs.PageCount

   'Если мы оказались здесь, значит, переменная rs не пустая,
   ↳поэтому выводим результаты.
   Response.Write("<TABLE BORDER=""0"" BORDERCOLOR=""ORANGE"">")


   Response.Write("<TR>")
   'Переписываем стиль шрифта,
   ↳потому что используем другой цвет
   Response.Write("<TD COLSPAN=""10""><FONT FACE=""arial,
   ↳verdana, helvetica"" SIZE=""2"" COLOR=""#ff8c00"">")
   Response.Write("Your search returned "
   ↳ & rs.RecordCount & " records and you are on page"
   ↳ & iPageCurrent & " of " & iPageCount)
   Response.Write("</FONT></TD>")
   Response.Write("</TR>")

   'Сокрытие текущей страницы
   Response.Write("<INPUT TYPE=""HIDDEN"" NAME=""PageCurrent""
   ↳VALUE=" & """" & iPageCurrent & """" & ">")
   Response.Write("<TR>")
   Response.Write("<TD COLSPAN=""4"">")
   If iPageCurrent >1 Then
      Response.Write("<INPUT TYPE=""Image""
      ↳ALT=""Previous Records""
      ↳SRC=""Images/Previous.gif"" VALUE=""Previous""
      ↳NAME=""Previous"" WIDTH=""54"" HEIGHT=""10""> ")
   End If

   If iPageCurrent < iPageCount Then
      Response.Write("<INPUT TYPE=""Image""
      ↳ALT=""Next Records""
      ↳SRC=""Images/Next.gif"" VALUE=""Next"" NAME=""Next""
      ↳WIDTH=""54"" HEIGHT=""10"">")
   End If

   Response.Write("</TD>")
   Response.Write("</TR>")

```

```
    'Заголовок таблицы
    Response.Write("<TR>")
    Response.Write("<TD>" & sFontStyle & "<B>Person's
    ➥Name</B></FONT></TD>")
    Response.Write("<TD>" & sFontStyle & "<B>Date
    ➥of Birth<B></FONT></TD>")
    Response.Write("<TD>" & sFontStyle & "<B>Phone
    ➥No.</B></FONT></TD>")
    Response.Write("</TR>")

    Do While rs.AbsolutePage = iPageCurrent And Not rs.EOF
       Response.Write("<TR>")
       Response.Write("<TD>" & sFontStyle & rs("FullName ")
       ➥ & "</FONT></TD>")
       Response.Write("<TD>" & sFontStyle & rs("DOB")
       ➥ & "</FONT></TD>")
       Response.Write("<TD>" & sFontStyle & rs("PhoneNo")
       ➥ & "</FONT></TD>")
       Response.Write("</TR>")
       rs.MoveNext
    Loop
    Response.Write("</TABLE>")

    'Не забудьте все очистить
    rs.Close
    Set rs = Nothing
    Set Cmd = Nothing
End Sub%>
```

Обратите внимание на то, как в программе использованы возможности свойств объекта Recordset. С их помощью в программе задается, что на экране одновременно могут быть отображены только 10 записей. Зачем это сделано? Если на экране будет более 10 записей, то в окне появится полоса прокрутки. Лучше обойтись без нее, она выглядит некрасиво.

Итак, господа, у нас есть базовый Web-узел (с весьма изысканной графикой) и мы можем выполнять поиск в таблице Person.

Как вы уже догадались, страница Search.asp расположена в корне Web-проекта.

Теперь вы можете расширять возможности программы, например выполнять поиск в нескольких таблицах или возвращать еще и адреса подозреваемых. Однако перед этим рассмотрим некоторые вопросы практической реализации Web-узла.

Наблюдение за скоростью выполнения и целостностью данных

Главный вопрос при разработке Web-узла — быстродействие. Нет ничего хуже, чем ожидание загрузки Web-страницы. Ведь 15 секунд перед экраном компьютера ощущаются как целые 15 минут! Для повышения быстродействия придерживайтесь нескольких правил.

- Делайте страницы небольшими. Слишком много строк программы (более 400) скорее всего приведут к снижению быстродействия.
- Не переключайтесь часто между HTML и VBScript — это замедляет выполнение.

- Используйте объект Command. Узел можно построить и без него, однако работать он будет медленнее, потому что ADO должен знать, что вы собираетесь передавать базе данных.
- И наконец, делайте хранимые процедуры как можно более эффективными (при тех же результатах).

Работа с узлом SQLSpyNet

Теперь посмотрим, как работает наш узел. Для этого выполните следующее.

1. Откройте окно броузера и введите в поле адреса имя вашего сервера в таком виде: **http://grunter/SpyNet**. Экран будет выглядеть, как на рис. 12.5.
2. Введите имя пользователя SpyNetIntranetUser и пароль. Щелкните на кнопке LOGIN. Экран примет вид, как на рис. 12.6.
3. В правой верхней части экрана есть кнопка SEARCH (а также HOME и LOGOUT). Щелкните на кнопке SEARCH. Экран примет вид, как на рис. 12.7. Здесь можно вводить условия поиска.
4. Введите буквы и строки в поля ввода и щелкните на кнопке GO (Выполнить). Я ввел строку Greg, и на экране появился результат, показанный на рис. 12.8.

Итак, Web-узел работает! Этот интерфейс довольно прост, но может служить хорошим фундаментом для построения более совершенных конструкций.



Рис. 12.5. Страница Login.asp, ожидающая ввода имени пользователя и пароля

Рис. 12.6. Страница `Welcome.asp`, приветствующая пользователя после успешной регистрации

Рис. 12.7. Страница `Search.asp` для ввода условий поиска и возвращения его результатов



Рис. 12.8. Получение данных, удовлетворяющих условиям поиска

Резюме

В этой главе было рассмотрено создание базового Web-узла, ко доставлять пользователям. Но для раскрытия всех возможносте практиковаться, практиковаться и практиковаться! В исходные SQLSpyNet включены комментарии; внимательно изучите их и ченные знания на практике.

Следующие шаги

В следующей главе будет рассмотрен мастер DTS (Data Trans предоставляющий множество гибких средств для импорта данн Данные будут импортироваться из электронных таблиц Excel, кот чить на Web-узле Издательского дома "Вильямс" (www.williamspub

Глава 13

Сбор разрозненных данных в один источник

В этой главе...

В предыдущих главах были рассмотрены основы разработки баз данных SQL Server 2000 и реализован простой Web-интерфейс, позволяющий пользователям читать и просматривать данные. Вы получили столько информации, что теперь, возможно, у вас от нее кружится голова! Ну а сейчас сядьте поудобнее и расслабьтесь: эта глава легкая и приятная.

Мы не рассмотрели абсолютно все, что есть в SQL Server 2000, но вы получили достаточное представление о его базовых компонентах и о том, как работать с интерфейсом. В этой главе мы подробнее разберемся с нашими тайными агентами и злоумышленниками, а также с новой информацией, ожидающей представления.

До сих пор наши данные были довольно ограниченными и простыми. Теперь приступим к импортированию более сложных данных из электронных таблиц Excel 2000, воспользовавшись для этого средством преобразования данных DTS (Data Transformation Service).



На заметку Импортировать данные можно не только из электронных таблиц Excel 2000. Сервер SQL Server 2000 может импортировать данные почти из любой версии Excel. Однако, чтобы вы могли воспользоваться подготовленными данными, пусть это будет формат Excel 2000.

Электронная таблица может быть источником имен тайных агентов, получаемых из разведывательных организаций всего мира, или электронной копией информации из газет, которая собиралась годами.

В этой главе речь пойдет также о расширении нашего приложения. Вы готовы к этому? Но успокойтесь. Я шучу. Для одной книги расширений уже более чем достаточно, это было бы слишком жестоко с моей стороны. В следующих двух главах вы будете оттачивать мастерство администратора баз данных, изучая некоторые дополнительные средства, предоставляемые SQL Server 2000.

Занесение новых данных в базу данных SQLSpyNet с помощью мастера DTS

Мастер DTS — чрезвычайно гибкий инструмент, позволяющий импортировать данные почти из любых источников, включая Excel, DB2, Oracle, Access и даже текстовые файлы. Зачем это нужно? Во многих организациях данные разбросаны по всем отделам. У Чарльза данные о клиентах хранятся в электронных таблицах, у Мэри все поставщики содержатся в небольшой базе данных Access, а в секретариате в текстовых файлах (или файлах Word) хранятся имена и телефонные номера сотрудников компании.

Работа с данными в таком виде сродни ночному кошмару! Неизбежным становится дублирование данных, а их получение чрезвычайно затруднено.

Но здесь появляется "служба спасения" — SQL Server 2000. С помощью программы DTS, входящей в состав SQL Server еще с версии 7.0, организации смогут наконец собрать разрозненные куски информации воедино. Мастер DTS предназначен для импорта (или экспорта) в базу данных по принципу "прицелься и стреляй". В этой главе мы пройдем все этапы работы мастера, включая выбор базы данных для импорта, задание источника данных и т.д.

Все это хорошо для простых данных, но как быть с данными, отношения в которых сложнее, чем отношение имени и фамилии? Программа DTS способна справиться и с такими данными (с вашей помощью, конечно).

В пакете DTS можно определить сценарий VBScript, который будет использован для обработки и форматирования данных. Этот сценарий предоставляет большие возможности изменения данных перед их вставкой в базу данных, в процессе и после вставки.

О пакетах DTS можно сказать очень многое, однако рассмотрим лишь основы их применения. Их гибкость и возможности управления процессами импорта/экспорта приятно удивят вас, и я уверен, в SQL Server 2000 вы обязательно будете использовать пакеты DTS.

Прежде чем приступить к вставкам в базу данных, удалим из нее *все* данные. Начинать с чистого листа всегда легче. Однако перед этим создайте резервную копию базы данных.

На заметку Создание резервной копии перед большими изменениями данных должно войти у вас в привычку.

Резервное копирование базы данных перед переносом

Вам, конечно же, лень заниматься этим. Однако сообщу хорошую новость: вам не придется писать операторы резервного копирования заново! В свое время мы уже создали задачу резервного копирования (глава 11, "Администрирование разведывательной сети"), поэтому сейчас можем просто выполнить ее. Другими словами, чтобы создать полную резервную копию базы данных, достаточно щелкнуть правой кнопкой мыши на задаче (в папке Management\SQL Server Agent\Jobs) и выбрать команду Start Job (рис. 13.1).

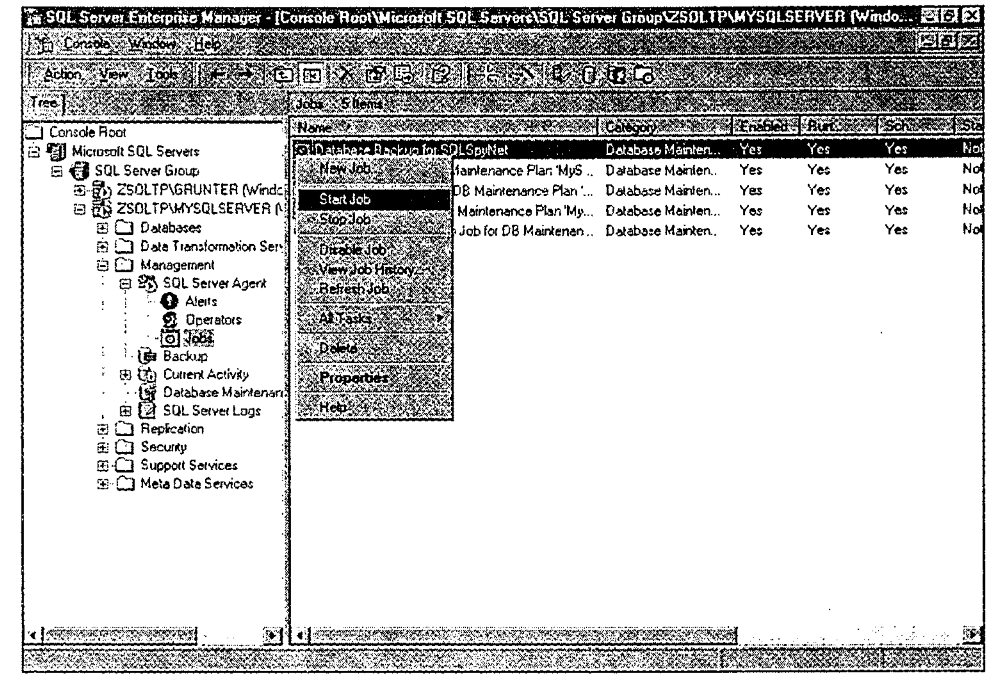


Рис. 13.1. Запуск задачи в Enterprise Manager

Вы увидите состояние задачи Executing (Выполняется). По завершении задачи ее состояние изменится на Succeeded (Успешно) или Failed (Неуспешно).

Итак, резервное копирование выполнено просто и быстро, даже без ввода команд!

Создание хранимой процедуры для очистки данных

Теперь создадим хранимую процедуру, которая удалит из таблиц все данные.

На заметку При использовании DTS удалять все данные не обязательно, однако, как вы помните, мы хотим начать с чистого листа. Лучший способ сделать это — воспользоваться хранимой процедурой, удаляющей данные.

Мы создаем хранимую процедуру, выполняющую операторы DELETE, потому что, возможно, захотим сделать это еще раз. Но сейчас сделаем это несколько иначе: создадим хранимую процедуру с помощью средства Enterprise Manager. А теперь защитим хранимую процедуру таким образом, чтобы ее мог выполнить только член роли dbo. Ведь мы не хотим, чтобы каждый пользователь мог удалить все данные из нашей базы данных!

Создание хранимой процедуры с помощью средства *Enterprise Manager*

Для создания хранимой процедуры, удаляющей все данные из базы данных SQLSpyNet, выполните ряд действий.

1. Запустите Enterprise Manager. В папке SQLSpyNet найдите папку Stored Procedures.
2. Щелкните правой кнопкой на папке Stored Procedures и из появившегося контекстного меню выберите команду New Stored Procedure (рис. 13.2).

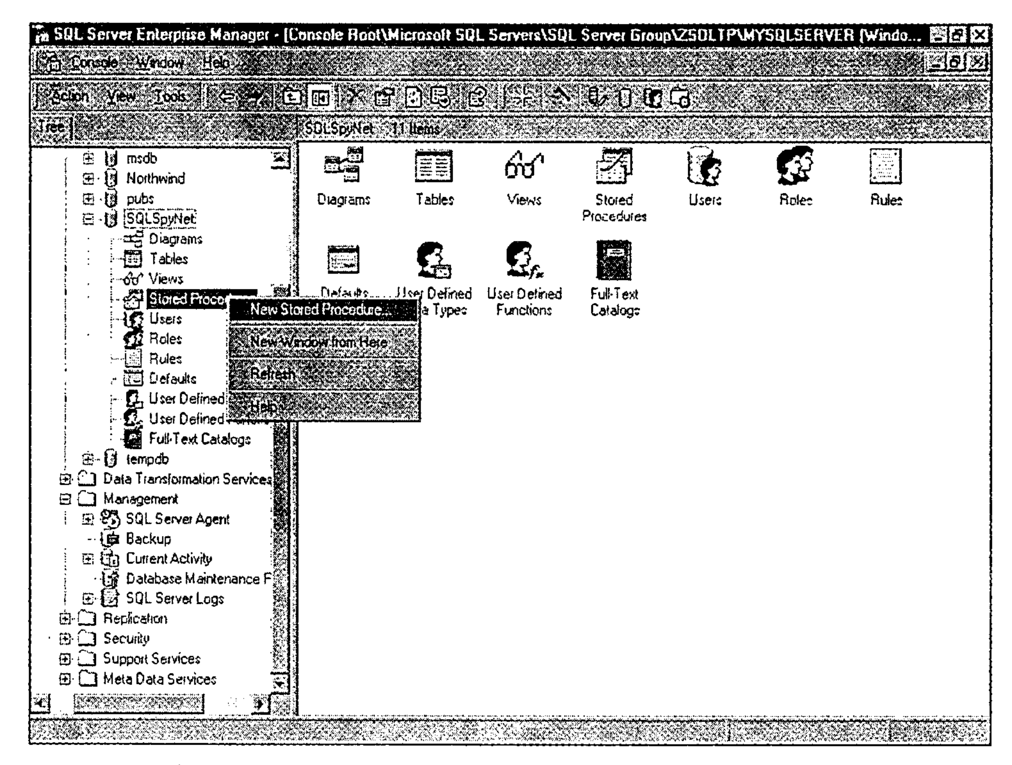


Рис. 13.2. Создание в базе данных SQLSpyNet хранимой процедуры с помощью Enterprise Manager

Появится диалоговое окно создания хранимой процедуры с шаблоном процедуры (рис. 13.3).

Экскурс

Более трудный способ программирования

Вы можете спросить: "Зачем писать текст программы? Ведь в *Enterprise Manager* это можно сделать с помощью шаблонов".

Во-первых, если вы умеете писать текст хранимой процедуры "с нуля" (а вы должны уметь это делать), то для вас не составит труда создать ее в *Enterprise Manager*. Во-вторых, в написанном тексте значительно проще найти ошибки, чем в тексте, созданном с помощью шаблона. Кроме того, в интерфейсе может проявиться ошибка, и вам все же придется писать и анализировать текст. Помните: мастерство достигается практикой.

3. Теперь присвойте создаваемой хранимой процедуре имя, напоминающее о том, что она делает, например *NukeAllData*.
4. В окне New Stored Procedure введите код листинга 13.1.

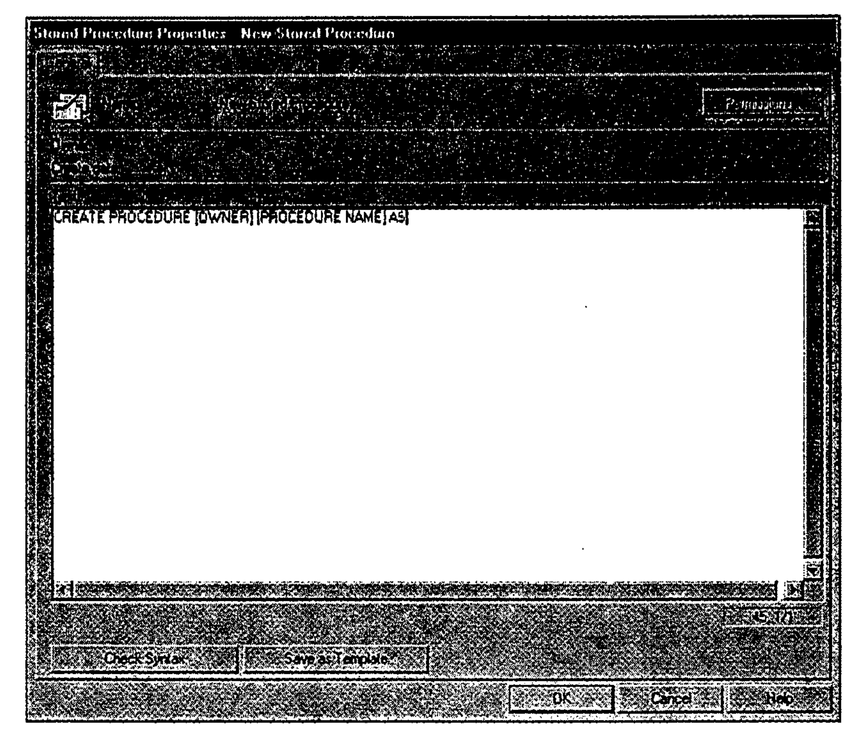


Рис. 13.3. Диалоговое окно создания хранимой процедуры с шаблоном (оператор CREATE PROCEDURE*)*

Листинг 13.1. Создание хранимой процедуры удаления данных

Код для запуска →

```
CREATE PROCEDURE NukeAllData
   @Confirm INT = 0
AS
   IF @Confirm <> 0
      BEGIN
         BEGIN TRANSACTION
         DECLARE @LocalError INT
         SET @LocalError = 0
         DELETE FROM Activity
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM ActivityType
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM BadGuy
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM Spy
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM Address
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM Person
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM AddressType
         SET @LocalError = @LocalError + @@ERROR
         DELETE FROM Country
         SET @LocalError = @LocalError + @@ERROR
```

```
            IF @LocalError <> 0
               ROLLBACK TRANSACTION
            ELSE
               COMMIT TRANSACTION
         END
      ELSE
         PRINT 'You cannot execute this
         ↳Stored Procedure at this time'
GO
```

Анализ

Что делает эта хранимая процедура? Она удаляет *все* данные из *всех* таблиц нашей базы данных. Все ее действия включены в одну транзакцию на случай, если что-нибудь не сработает.



Помните: удалить родительские записи, не удалив предварительно всех дочерних, невозможно.

Параметр, передаваемый в хранимую процедуру, должен быть отличен от нуля. Он предназначен для того, чтобы случайный запуск процедуры не удалил все данные.

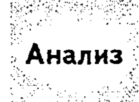
Ограничение прав выполнения процедуры удаления

Далее следует отменить право выполнения созданной процедуры для других пользователей и ролей базы данных. Члены учетной записи sa и роли dbo по-прежнему смогут выполнять ее. Для отмены прав доступа введите код листинга 13.2 в окне Query Analyzer.

Листинг 13.2. Защита хранимой процедуры

Код для запуска

```
DENY EXECUTE ON NukeAllData TO SQLSpyNetUser
GO
DENY EXECUTE ON NukeAllData TO SpyNetIntranetUser
GO
DENY EXECUTE ON NukeAllData TO SQLSpyNetRole
GO
DENY EXECUTE ON NukeAllData TO Public
GO
```



Оператор DENY отменяет право пользователя (или роли) выполнять хранимую процедуру. Фактически его действие обратно действию оператора GRANT. Другими словами, вместо того чтобы предоставить право одним пользователям, мы отменили право других пользователей.

Выполнение хранимой процедуры

Теперь для удаления всех данных из нашей базы данных необходимо выполнить хранимую процедуру. Введите код листинга 13.3 в окне Query Analyzer.

Листинг 13.3. Удаление данных

Код для запуска

```
EXEC NudeAllData 1
```

На заметку

Если попытаться выполнить процедуру, не передав ей параметр, то процедура ничего не сделает и возвратит строку с сообщением, записанным в ее исходном тексте. Попробуйте сделать это.

Если вас заинтересует, что означает 1, посмотрите в текст процедуры, и вы все поймете.

Итак, мы создали резервную копию базы данных, удалили все данные и теперь готовы импортировать данные в наше приложение.

Загрузка базы данных SQLSpyNet с помощью мастера DTS

Создадим пакет DTS, который сможет подарить новые данные о наших доблестных разведчиках и злоумышленниках.

На заметку

С помощью мастера DTS процесс импорта данных фактически выполняется трижды: сначала для первичных таблиц, затем для вторичных и наконец для таблицы `Activity`.

Не забудьте загрузить с Web-узла Издательского дома "Вильямс" материалы к этой книге. В них вы найдете три файла: `SpyDatabasePrimaryTables.xls`, `SpyDatabaseSecondaryTables.xls` и `SpyDataActivityTable.xls`.

Зачем выполнять импорт три раза? Как вы знаете, дочерние записи базы данных не могут существовать без соответствующих родительских записей. Первый импорт добавляет родительские записи. После этого может выполняться второй импорт, потому что необходимые записи уже есть. Третий импорт может выполняться потому, что есть уже все необходимые записи.

Для импорта первичных таблиц выполните в мастере приведенную ниже пошаговую инструкцию. Позже в этом разделе вы предпримете эти же шаги для импорта оставшихся таблиц.

1. Щелкните на папке Data Transformation Services правой кнопкой и выберите команду All Tasks⇨Import Data (рис. 13.4).

   Запускается мастер DTS Import/Export Wizard, в котором мы зададим различные опции процесса импорта. Начальное окно мастера показано на рис. 13.5.



Это не единственный способ запуска мастера DTS. Как вы уже знаете, SQL Server 2000 поддерживает различные способы запуска. Мастер DTS можно запустить из главного меню (команда *Start⇨Programs⇨Microsoft SQL Server⇨Import and Export Data*), а также в окне *Enterprise Manager* из меню *Tools⇨Data Transformation Service⇨Import Data* (или *Export Data*).

Рис. 13.4. Запуск мастера преобразования DTS в Enterprise Manager

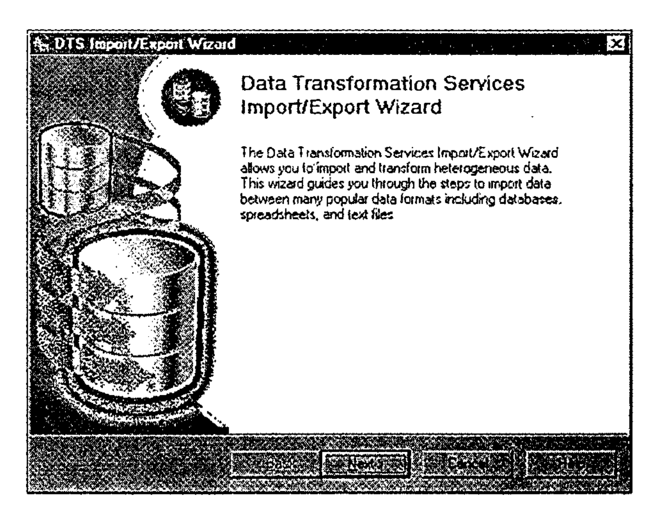


Рис. 13.5. Мастер DTS Import/Export Wizard готов к работе

2. Как почти во всех мастерах, в первом окне ничего не нужно делать. Щелкните на кнопке Next.

3. Следующее окно (рис. 13.6) более интересное. В нем можно задать источник, из которого мы хотим считывать данные.

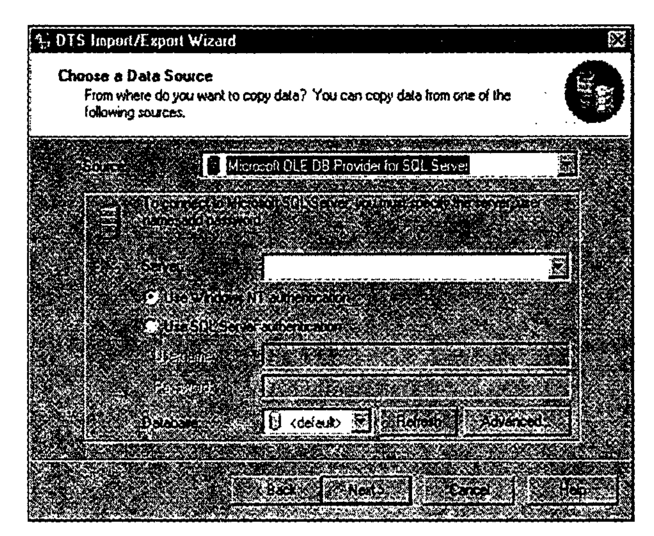


Рис. 13.6. Мастер DTS Import/Export Wizard готов к определению источника, из которого мы хотим выбрать данные

В раскрывающемся списке Source (Источник) можно указать, откуда доставляются данные. Сейчас выберите Microsoft Excel 97-2000. При этом в окне останется только поле ввода имени файла (рис. 13.7).

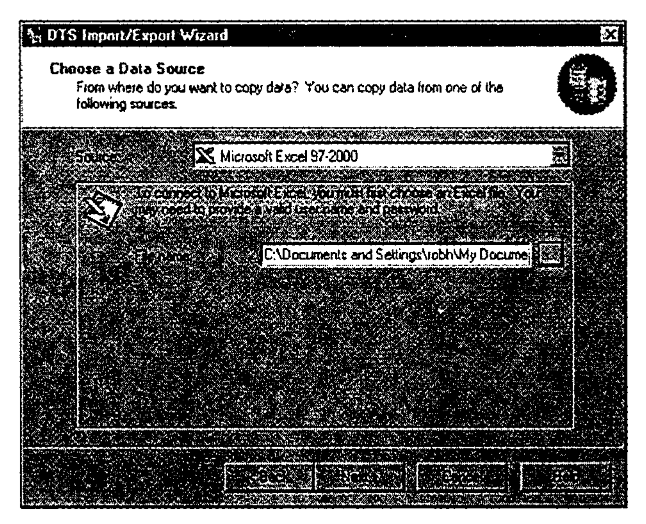


Рис. 13.7. Мастер DTS Import/Export Wizard готов к заданию имени и маршрута файла, содержащего электронную таблицу Excel

4. Вы можете ввести имя файла с маршрутом или, щелкнув на кнопке с троеточием (...), найти нужный файл. После этого щелкните на кнопке Next.
5. В следующем окне можно указать источник данных, *куда* мы хотим импортировать данные (рис. 13.8). Это может быть база данных SQL Server 2000 или почти любой другой источник.



Для импорта/экспорта можно задавать источники данных, отличные от SQL Server 2000, а значит, можно переносить данные в электронные таблицы Excel или из них, а также практически между любыми источниками данных. Программа DTS выступает во всем процессе лишь как посредник.

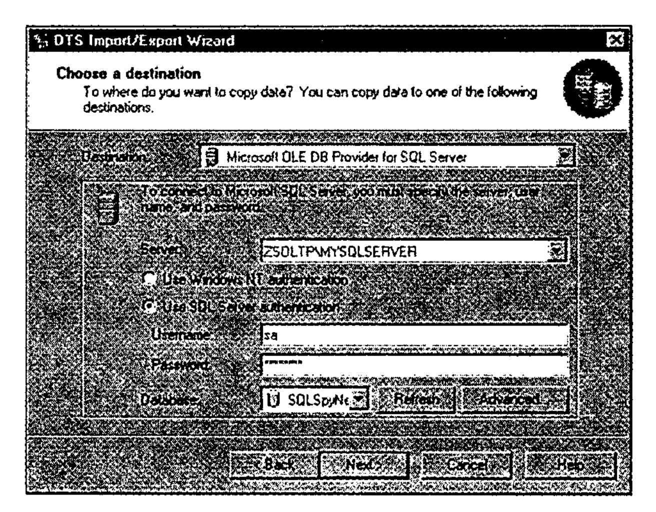


Рис. 13.8. Установка в мастере DTS Import/Export Wizard опций адресата данных

В этом окне мастера нужно задать следующие опции адресата данных:

- в раскрывающемся списке Destination (Адресат) выберите Microsoft OLE DB Provider for SQL Server;
- в раскрывающемся списке Server выберите свой сервер;
- установите флажок Use SQL Server authentication; в поле Username введите **sa**; в поле Password — легко запоминаемый пароль;
- в раскрывающемся списке Database выберите базу данных SQLSpyNet. Если в этом списке ничего нет, щелкните на кнопке Refresh (Обновить).

6. Установив указанные параметры, щелкните на кнопке Next.



С помощью кнопки *Advanced* (Дополнительно) можно установить более сложные опции, например максимальную продолжительность соединения с адресатом данных.

7. В следующем окне (Specify Table Copy or Query) можно указать, что вы хотите копировать все таблицы или выполнить для копирования собственный запрос. Мы копируем из электронной таблицы Excel, поэтому установите переключатель Copy table(s) and view(s) from the source database (Копировать таблицы и представления из базы данных — источника), как показано на рис. 13.9.
8. Щелкните на кнопке Next. Появится окно, показанное на рис. 13.10.

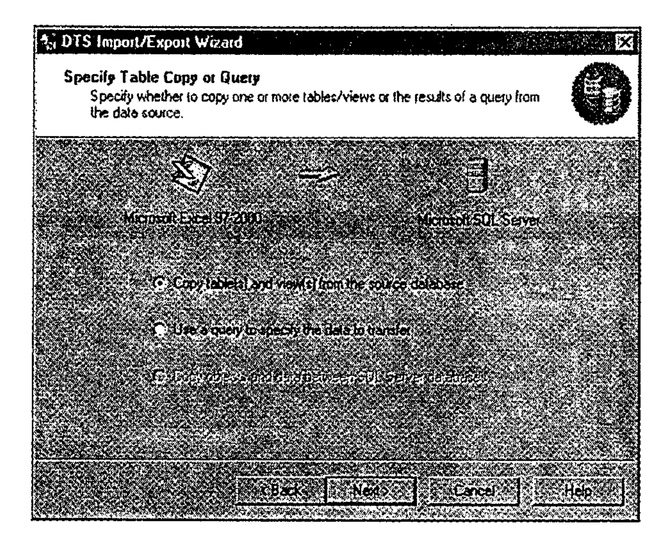


Рис. 13.9. Задание типа выполняемого импорта

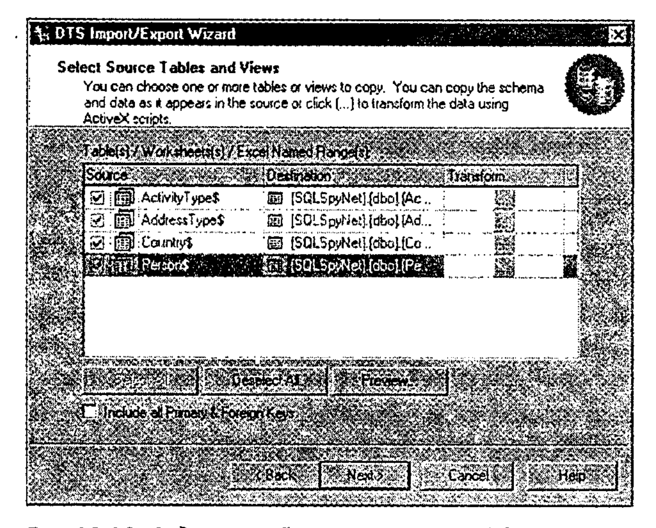


Рис. 13.10. Задание таблицы, в которую будут копироваться данные

9. В этом окне можно задать таблицы и столбцы, в которые мы хотим копировать данные. Установите следующее соответствие между таблицами:
   - лист ActivityType — таблица `ActivityType`;
   - лист AddressType — таблица `AddressType`;
   - лист Country — таблица `Country`;
   - лист Person — таблица `Person`.
10. Не устанавливайте флажок Include all Primary & Foreign Keys (Включить все первичные и вторичные ключи).



Если к моменту начала импорта таблиц не существовало, эта опция создаст для них первичные и вторичные ключи.

11. Установив соответствие таблиц, щелкните на кнопке Next.
12. С помощью окна, показанного на рис. 13.11, можно задать немедленное выполнение импорта (Run immediately) или создать расписание для его последующего выполнения (Schedule DTS package for later execution). Мы будем выполнять импорт сейчас (чуть позже), поэтому установите флажок Run immediately.

    Если установить флажок Save DTS Package (Сохранить пакет DTS), пакет сохраняется и его можно будет впоследствии выполнить. Если сохранить пакет, то его можно не только выполнять позже, но и настраивать в соответствии с возникающими требованиями.

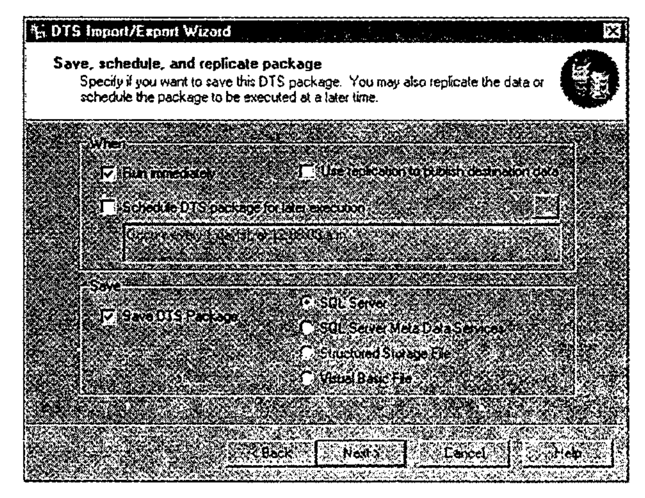


Рис. 13.11. Установка опций выполнения и сохранения пакета



Если флажок *Save DTS Package* снят, то установите флажок *Schedule DTS package for later execution*, а затем снова снимите. Эти действия приведут к активизации флажка *Save DTS Package*.

13. С параметром Save DTS Package связано несколько переключателей.
    - SQL Server. Пакет сохраняется в базе данных msdb в таблице stsdtspackages.
    - SQL Server Meta Data Services. Пакет сохраняется в виде Meta Data Services.
    - Structured Storage File. Позволяет сохранить пакет в виде COM-файла.
    - Visual Basic File. Позволяет сохранить пакет в виде файла Visual Basic.
14. Установите переключатель SQL Server и щелкните на кнопке Next.
15. Появляется окно (рис. 13.12), с помощью которого можно выбрать имя сохраняемого пакета. Как обычно, нужно ввести имя пакета и описание. Описание должно быть достаточно информативным, чтобы из него можно было понять, что делает пакет. Опции этого окна имеют следующее назначение.

- Name. Имя пакета. Введите **SQL Spy Net Excel Primary Tables Import**.
- Description. Описание. Введите **Импорт электронных таблиц Excel; заполнение первичных таблиц; выполнил *имя*; дата создания *дата*.**
- Owner password. Для защиты пакета в этом поле можно ввести пароль его владельца. Оставьте поле пустым.
- User password. В этом поле можно ввести пароль пользователя. Это значит, что пользователь сможет выполнять пакет, но не сможет просматривать его содержимое. Оставьте поле пустым.
- Server name. Используя имя и пароль пользователя, введите в этом поле имя сервера, в котором находится база данных SQLSpyNet (на этом сервере будет сохранен пакет DTS).

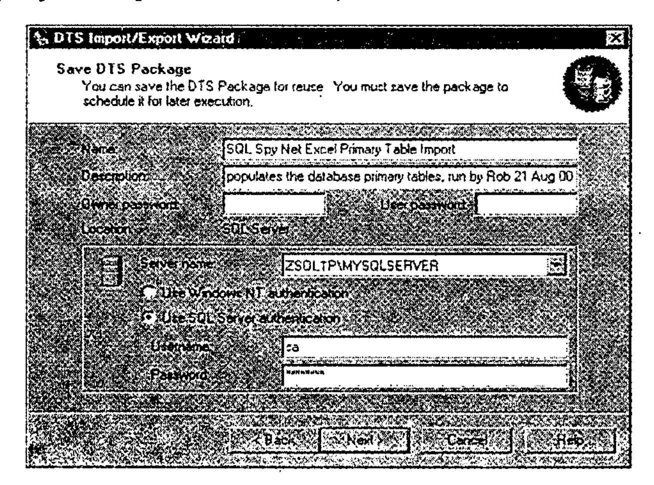


Рис. 13.12. Выбор параметров сохранения пакета



В раскрывающемся списке можно выбрать другой сервер (если он есть), в котором будет сохранен пакет. Сервер, отличный от того, с которым сейчас установлена связь, называется удаленным. Для доступа к нему и сохранения в нем пакета DTS необходимо иметь соответствующую учетную запись и знать пароль. Таким образом, можно создать пакет DTS на своем компьютере, а затем сохранить его на другом сервере для дальнейшего использования или в качестве резервной копии.

16. Установив перечисленные опции, щелкните на кнопке Next.
17. Завершающее окно мастера показано на рис. 13.13.
18. Если все установлено правильно, щелкните на кнопке Finish. При этом пакет DTS будет выполнен немедленно и данные будут импортированы. Вы увидите окно, показанное на рис. 13.14.

Когда мастер выполнит импорт основных (родительских) таблиц, нужно импортировать дополнительные (дочерние) таблицы. Для создания нового пакета запустите мастер еще раз. Теперь импортируется файл `SpyDatabaseSecondaryTables.xls`. Установите следующее соответствие таблиц:

- лист Spy — таблица Spy;
- лист BadGuy — таблица BadGuy;
- лист Address — таблица `Address`.

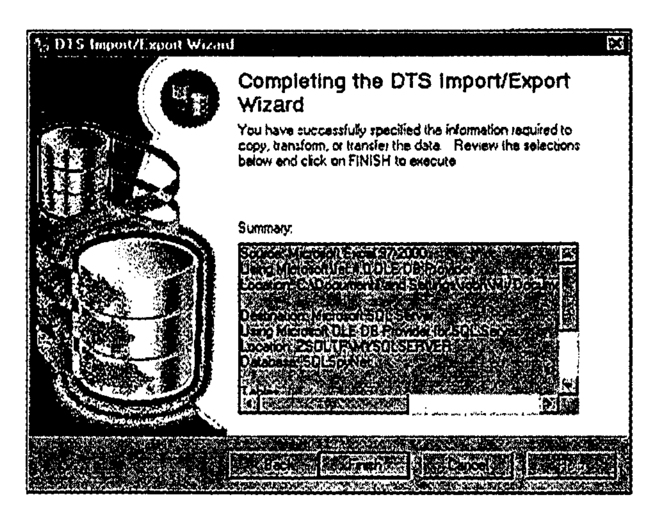


Рис. 13.13. Завершающее окно мастера DTS Import/Export Wizard

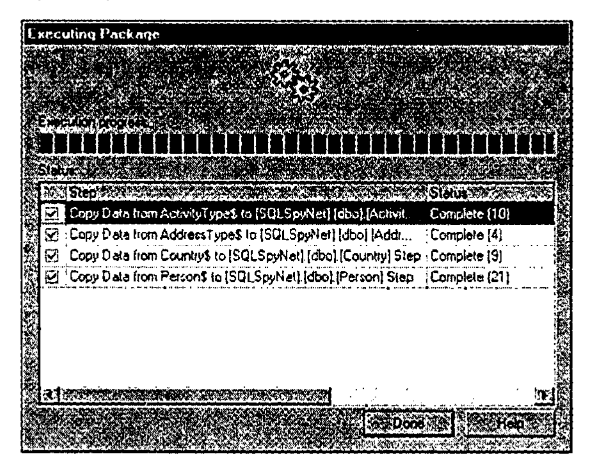


Рис. 13.14. Выполнение пакета мастером DTS Import/Export Wizard

Сохраните пакет DTS со следующими опциями:

- Name — SQL Spy Net Excel Secondary Tables Import;
- Description — Импорт электронных таблиц Excel; заполнение вторичных таблиц базы данных; выполнил Роб 30 августа 2001 года.

Теперь (извините за повторение) запустите мастер еще раз и создайте новый пакет DTS. На этот раз импортируйте файл `SpyDatabaseActivityTable.xls` и, как и раньше, установите соответствие листов Activity в Excel и в базе данных SQLSpyNet.

Как и раньше, сохраните пакет DTS, установив следующие опции:

- Name — SQL Spy Net Excel Activity Table Import;
- Description — Импорт электронной таблицы Activity; заполнение таблиц Activity базы данных; выполнил Роб 30 августа 2001 года.

После выполнения этих трех пакетов необходимо проверить таблицы и убедиться, что данные импортированы правильно.

Проверка результатов импорта

Как проверить, есть ли вообще строки в нашей таблице? Для этого используйте функцию COUNT, возвращающую количество строк в таблице. Откройте окно Query Analyzer и введите код листинга 13.4.

Листинг 13.4. Проверка данных после импорта

Код для запуска

```
1: SELECT COUNT(PersonID) FROM Person
```

Запрос возвратит текущее количество строк (вернее, непустых полей PersonID) таблицы Person.

Таким же образом проверьте остальные таблицы. Все они должны быть непустыми.

Поскольку мы сохранили пакеты DTS, теперь можно посмотреть, что делает SQL Server 2000, создавая эти пакеты. В окне Enterprise Manager откройте папку Data Transformation Services\Local Packages. Вы увидите три новых пакета. Дважды щелкните на одном из них — появится графическое представление пакета (рис. 13.15).

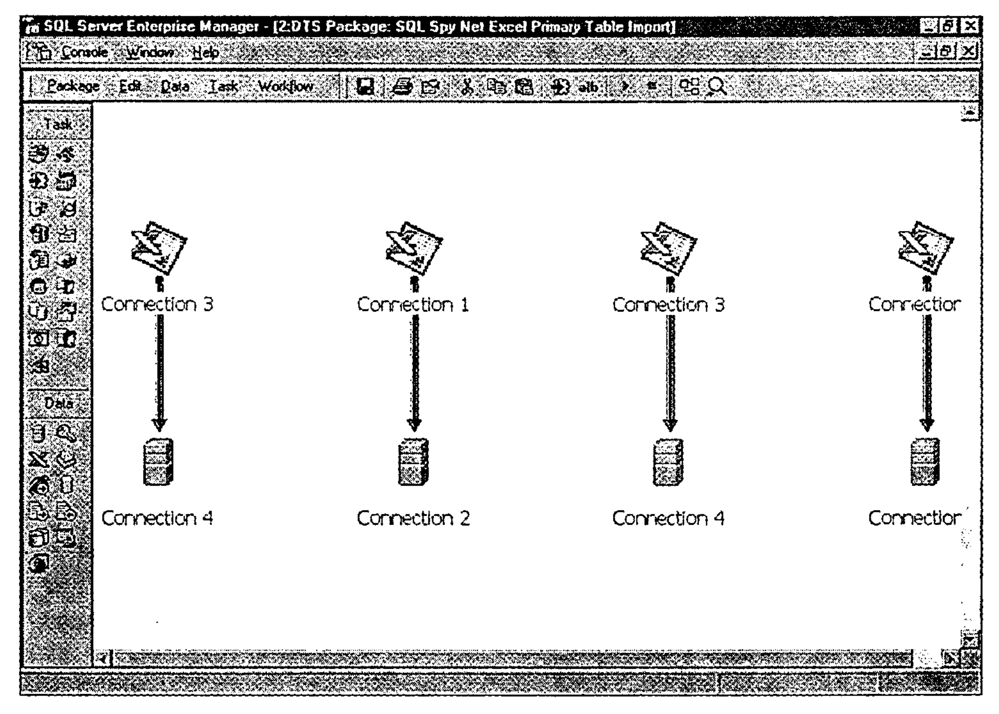


Рис. 13.15. Пакет DTS, сохраненный мастером DTS Import/Export Wizard

Сохраненный пакет можно изменить или выполнить повторно. Для просмотра опций импорта данных щелкните правой кнопкой мыши на какой-либо пиктограмме (включая стрелки) и в появившемся меню выберите команду Properties (рис. 13.16).

Рис. 13.16. Открытие окна свойств компонентов пакета DTS

В окне свойств можно изменять параметры пакета DTS. На рис. 13.17 показано окно свойств первого соединения (Connection1, одна из стрелок).

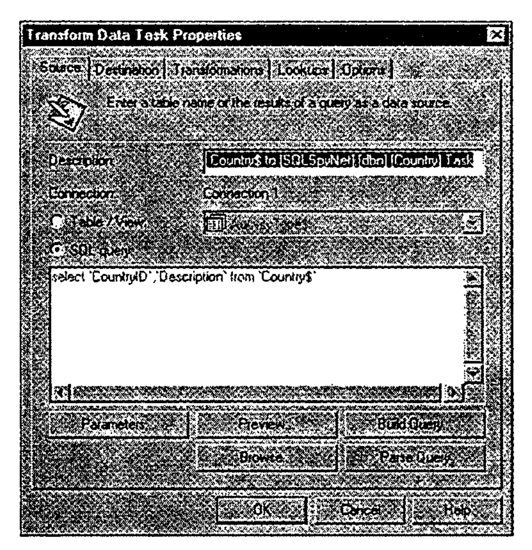


Рис. 13.17. Окно свойств первого соединения (Connection1) созданного пакета DTS

Таким образом, выполненные этапы мастера DTS Import/Export Wizard легко восстановить, поэтому с пакетами DTS можно смело экспериментировать.

Пусть это будет проверкой вашего мастерства! Сможете ли вы сконфигурировать импорт таким образом, чтобы все три группы электронных таблиц импортировались в одном пакете? Это вполне возможно, однако для этого понадобятся некоторые самостоятельные исследования и знания VBScript.

Что теперь делать с нашим приложением

Теперь наша база данных находится в стабильном состоянии. Первоначальной концепции базы данных не удалось избежать изменений, однако теперь возможности нашего приложения значительно шире.

Приложение SQLSpyNet предназначено для отслеживания передвижений тайных агентов и действий злоумышленников, поэтому логично будет добавить контроль оснащения агентов при выполнении заданий. Сейчас, когда наши агенты выходят на борьбу с "плохишами", мы не можем проконтролировать, насколько хорошо они оснащены. В фильме о Джеймсе Бонде мастер Кью постоянно докучает агенту 007 по поводу комплектности его оснащения. Не следует ли нам сделать так же?

Что для этого нужно изменить? Структура базы данных относительно стабильна, поэтому мы можем добавлять новые таблицы, не затрагивая существенно остальную базу данных. Понадобится добавить примерно две новые таблицы. Одна таблица (`Resource`) будет содержать список оборудования (ресурсы). Вторая таблица будет переходной, она предназначена для разрыва отношения типа "многие ко многим", между новой таблицей с оборудованием и таблицей `Activity`.

Как это стало возможным? Когда тайный агент выполняет задание, ему необходим некоторый ресурс (оборудование). Однако во время выполнения задания он может использовать больше одной единицы оборудования. Это создает между таблицами `Resource` и `Activity` отношение типа "многие ко многим", чем и объясняется необходимость переходной таблицы.

Реализовав эти изменения, что еще можно сделать с приложением? Следующий логичный шаг — использовать приложение как инструмент создания отчетов для руководства. Это позволит высшему руководству строить графики работы агентов и борьбы с злоумышленниками.

Мы можем также создавать отчеты (на основе Web или с помощью таких инструментов построения отчетов, как Crystal reports или Access) для контроля производительности, зарплат и списочного состава персонала. Разработать такой отчет довольно просто, достаточно создать некоторые хранимые процедуры (или представления) для чтения данных и разработать пользовательский интерфейс.

Возможны также следующие расширения нашего приложения.

- Добавление финансовых компонентов, позволяющих контролировать расходы. С их помощью можно определить, какая сумма тратится на борьбу с преступностью за год.
- Добавление в приложение других типов действующих лиц, кроме тайных агентов и злоумышленников.

Как видите, существует немало способов расширения нашего приложения, в результате чего заложенные в нем базовые возможности превзойдут все ожидания. Однако в реальной жизни первое, что мы должны будем сделать, — предоставить наше приложение пользователям. Когда пользователи получат доступ к приложению, мы, в свою очередь, получим сигнал обратной связи, информирующий нас о производительности, привлекательности и, главное, полезности нашего приложения.

Так что захватывающая работа над приложением далеко не закончена. Оно предоставит нам еще немало интересных проблем. И тогда дайте полную свободу своему воображению!

Резюме

По сравнению с другими главами книги, через которые мы "продирались" с таким трудом, эта глава самая легкая.

Мы рассмотрели создание пакетов DTS, предназначенных для импорта подробностей о наших агентах и злоумышленниках из электронных таблиц Excel. Могу с уверенностью предположить: теперь у вас не вызовет затруднений импорт данных из любых источников.

Затем мы рассмотрели текущее состояние нашего приложения и возможности его расширения. Однако помните: это далеко не все возможности. Не исключено, что у вас возникнут еще более замечательные идеи относительно его расширения и применения. И в этом случае дайте мне знать! Возможно, я напишу об этом другую книгу, и ваши идеи войдут в нее.

Следующие шаги

В следующей главе рассматривается отладка приложения. Вы узнаете, как с помощью инструментов SQL Server 2000 найти и устранить ошибки, возникающие при работе приложения.

Так что смените домашние тапочки на горные ботинки: мы выходим на ухабистую дорогу.

Глава 14

Отладка и устранение ошибок в SQL Server 2000

В этой главе...

Обычно в SQL Server 2000 все работает нормально, однако иногда все же возникают ошибки, которые нужно найти и устранить. Кроме того, если объем базы данных увеличивается, то обязательно возникнут проблемы с эффективным управлением ресурсами сервера.

В этой главе я поделюсь своим опытом решения этих проблем и попытаюсь дать общее представление о том, с чем вам придется столкнуться в реальной жизни.

Обнаружение ошибок с помощью средства *SQL Profiler*

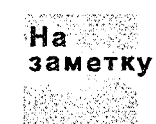
Инструмент SQL Profiler является одним из лучших средств SQL Server 2000 для решения возникающих проблем. С его помощью можно непосредственно увидеть, что было передано из клиентского приложения.

Последние несколько недель я был занят тем, что переносил базы данных SQL Server 6.5 на платформу SQL Server 7.0 (вы, конечно, удивитесь: почему не в SQL Server 2000, но желание заказчика — закон). У клиента был старый интерфейс Visual Basic (написанный кем-то когда-то), причем исходных текстов приложения у него не было. И здесь очень пригодился SQL Profiler. С его помощью я наблюдал операторы Transact-SQL, передаваемые в базу данных из клиентского приложения, и добивался их правильного функционирования. Инструмент оказался бесценным! Он помог перенести все необходимое настолько легко, насколько это вообще было возможно.

Теперь рассмотрим, что такое SQL Profiler и каковы его возможности. В этом разделе мы конфигурируем трассировку для перехвата команд, передаваемых базе данных SQL Server 2000 клиентским приложением SQLSpyNet.

Создание трассировки

Запустим трассировку и перехват *всех* команд, передаваемых броузером в базу данных SQLSpyNet, включая регистрационную информацию и поиск.



Возможность перехвата информации, поступающей на SQL Server из броузера, предоставляет один из лучших способов отладки. Способ этот прост и эффективен. Поэтому, если вы собираетесь разрабатывать пользовательские интерфейсы, уделите ему особое внимание.

1. Запустите SQL Profiler. Это можно сделать в Enterprise Manager (выберите команду Tools⇨SQL Server Profile) или из главного меню (выберите команду Start⇨Programs⇨Microsoft SQL Server⇨Profiler).
2. Когда SQL Profiler запущен, в первую очередь нужно установить определение трассировки. Для этого выберите команду File⇨New⇨Trace (рис. 14.1) или нажмите комбинацию клавиш <Ctrl+N>.

Рис. 14.1. Открытие окна определения новой трассировки

Сервер попросит ввести регистрационную информацию, чтобы можно было подтвердить соединение с базой данных, в которой мы хотим выполнять трассировку.

3. Заполните поля регистрации. Появится окно, показанное на рис. 14.2.
4. В этом окне нужно определить трассировку. Окно предоставляет несколько вкладок с несколькими опциями в каждой из них. Во вкладке General установите описанные ниже опции.
   - Trace name. Уникальное имя трассировки для ее распознавания в будущем. Введите **Trace for Spy Net**.
   - Trace SQL Server. Определяет экземпляр SQL Server, в котором выполняется трассировка. Если сейчас у вас больше одного экземпляра, укажите тот, который содержит базу данных SQLSpyNet.

- Template name. Позволяет выбрать один из предопределенных шаблонов, поставляемых с SQL Server 2000. Шаблоны помогают выполнять различные задачи администрирования баз данных. Например, с помощью шаблона SQLProfilerTuning можно настраивать запросы SQL и производительность базы данных. Выберите шаблон SQLProfilerStandard.
- Template file name. Позволяет выбрать шаблон, хранящийся в каком-либо файле, а не поставляемый с SQL Server 2000. Файлы шаблонов имеют расширение `.tdf`. Оставьте эту опцию по умолчанию.

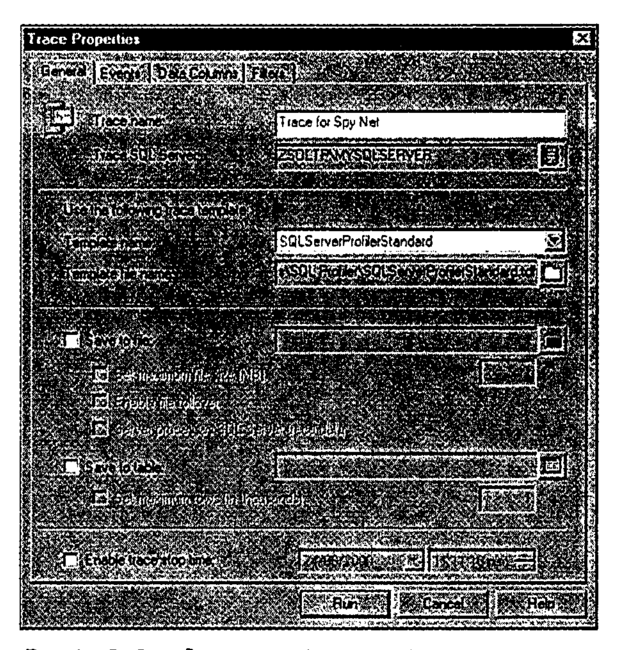
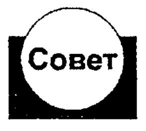


Рис. 14.2. Определение свойств новой трассировки

- Save to file. Задает имя файла, в котором сохраняется результат трассировки. Поэтому, если вы захотите выполнить аудит приложения (см. главу 9, "Обеспечение безопасности базы данных Spy Net"), можете перехватить все события сервера и базы данных, а затем сохранить файл трассировки на диске. Сейчас мы не будем этого делать.
- Save to table. Результат трассировки можно сохранить в таблице базы данных. Другими словами, перехваченные события можно сохранить в существующей или в новой таблице любой базы данных, к которой есть доступ. Сейчас нам это не нужно, оставьте опцию по умолчанию.
- Enable trace stop time. Задает продолжительность выполнения трассировки. Оставьте этот флажок снятым.



Последняя опция очень полезна для аудите приложения. С ее помощью можно включать трассировку в моменты пиковой нагрузки сети или для поимки потенциального хакера.

5. Активизируйте вкладку Events (События). Как показано на рис. 14.3, в ней можно задать классы событий, которые требуется отследить.

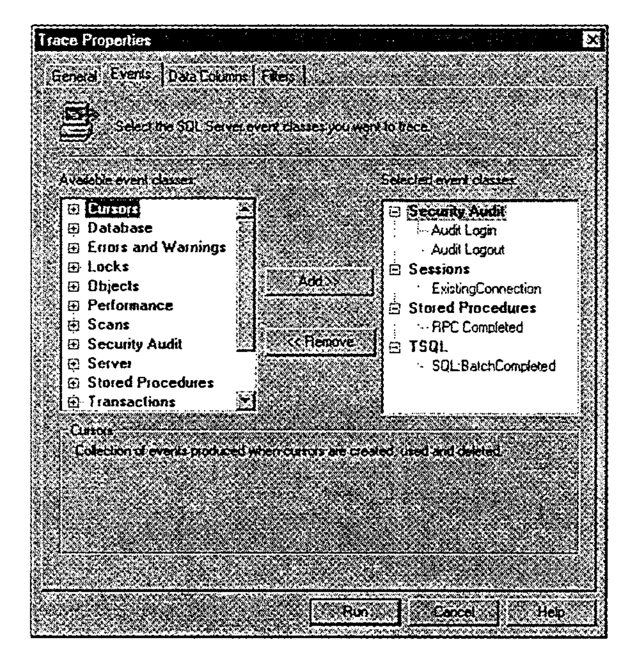


Рис. 14.3. Задание событий, которые должны быть перехвачены

С помощью классов событий можно перехватывать события основных категорий Transact-SQL в момент, когда они передаются, выполняются или завершаются. Например, можно перехватить запуск хранимой процедуры, ее перекомпиляцию или завершение. Однако сейчас оставьте опции шаблона SQLProfilerStandard по умолчанию.

6. Активизируйте вкладку Data Columns (рис. 14.4). В ней можно задать столбцы, в которых будет представлена информация о результатах трассировки.
7. Поскольку наш сервер работает под управлением операционной системы, отличной от семейства систем NT, удалите из поля Selected data (Выбранные данные) столбец NTUserName. Для этого выделите его и щелкните на кнопке Remove.



Если этот столбец не виден, разверните группу *Columns* (все группы выделены полужирным шрифтом).

8. Активизируйте вкладку Filters. Это последняя и самая полезная вкладка окна Profile Trace Properties. С ее помощью можно задать фильтры трассировки, которые ограничат перехватываемые события заданными базами данных, пользователями и даже приложениями.
9. Нас интересует только то, что происходит в базе данных SQLSpyNet. Поэтому найдите столбец DatabaseName, щелкните на знаке "плюс" (+) и в папке Like найдите критерий SQLSpyNet (рис. 14.5).

   Сконфигурированный таким образом фильтр пропустит только события, появляющиеся в базе данных SQLSpyNet.
10. Дело сделано! Теперь трассировка определена. Щелкните на кнопке Run, и трассировка запустится.

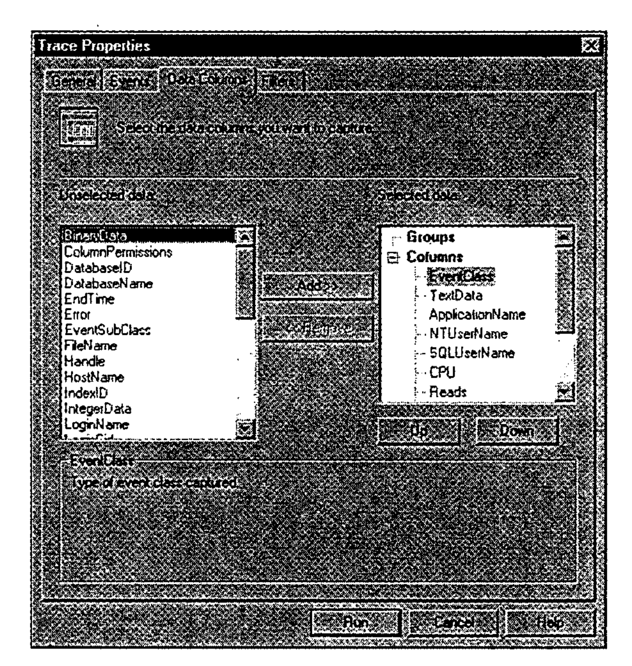


Рис. 14.4. Задание столбцов с результатами трассировки

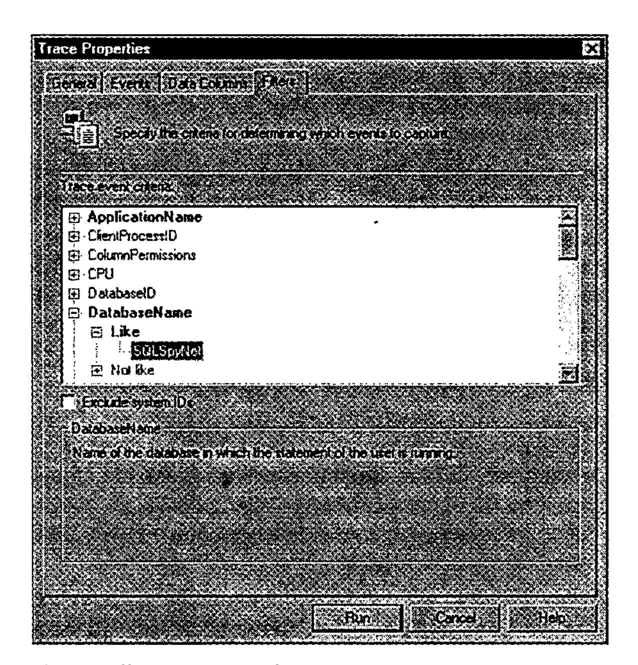


Рис. 14.5. Установка фильтров трассировки

Запущенная трассировка перехватывает основную информацию о существующих соединениях (если они есть), поэтому на экране вы увидите записи, которые покажутся вам не имеющими никакого отношения к приложению.



Экскурс

Установлено ли соединение?

Трассировка протоколирует информацию об установленном с SQL Server соединении. Это указывает на то, что пользователи уже подключены, поэтому выполнение большого запроса может помешать им.

Если трассировка указывает на существующее соединение (учитывая, что мы сейчас в однопользовательской среде), значит, у вас, видимо, осталось открытым окно *Enterprise Manager*, *Query Analyzer* или *Spy Net* (или все три вместе).

Чтобы очистить окно трассировки, нужно закрыть эти приложения, остановить трассировку (щелкнув на кнопке с красным квадратиком на панели инструментов) и очистить окно трассировки (выбрав команду меню *Edit⇨Clear Trace Window*). Запустите трассировку заново (щелкнув на кнопке с зеленым треугольником на панели инструментов), и вы увидите, что информация о соединениях отсутствует.

Наблюдение за пользователями

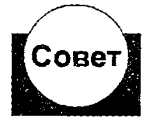
Итак, трассировка запущена. Как посмотреть, что делают пользователи?

На заметку

Прежде чем выполнять следующие шаги этого раздела, убедитесь, что окно SQL Profiler по-прежнему открыто и трассировка выполняется.

1. Активизируйте Web-интерфейс приложения SQLSpyNet, введите регистрационную информацию и активизируйте окно поиска.
2. Выполните поиск. На рис. 14.6 видно, что я выполнял поиск слова *Greg*. Соответствующая запись появилась в окне трассировки.

   Из этой записи видно, что пользователь (т.е. я) выполнил хранимую процедуру `PersonSearch` с параметрами `'Greg'` и `NULL`.
3. Эту команду можно скопировать в буфер обмена и выполнить в окне Query Analyzer, тогда вы увидите возвращаемый SQL Server 2000 результат запроса.



Можно также скопировать команду, переданную SQL Server клиентским приложением, щелкнув на ней и выделив текст в области *Results*. После этого щелкните правой кнопкой мыши и из появившегося контекстного меню выберите команду *Copy* или нажмите комбинацию клавиш <Ctrl+C>.

Как же использовать описанную возможность в целях отладки? Таким образом можно просматривать команды, передаваемые SQL Server клиентским приложением. Поэтому вы можете легко обнаружить пропущенные параметры, проверить, правильно ли они сформатированы, и т.д. Допустим, клиентское приложение постоянно возвращает сообщение об ошибке и вы не можете понять, откуда оно взялось. В этом случае перехватите команду, передайте SQL Server с помощью средства Query Analyzer, и в девяти случаях из десяти проблема будет решена.

Теперь, когда вы освоили основы работы со средством SQL Profiler, рассмотрим другие инструменты устранения возможных проблем.

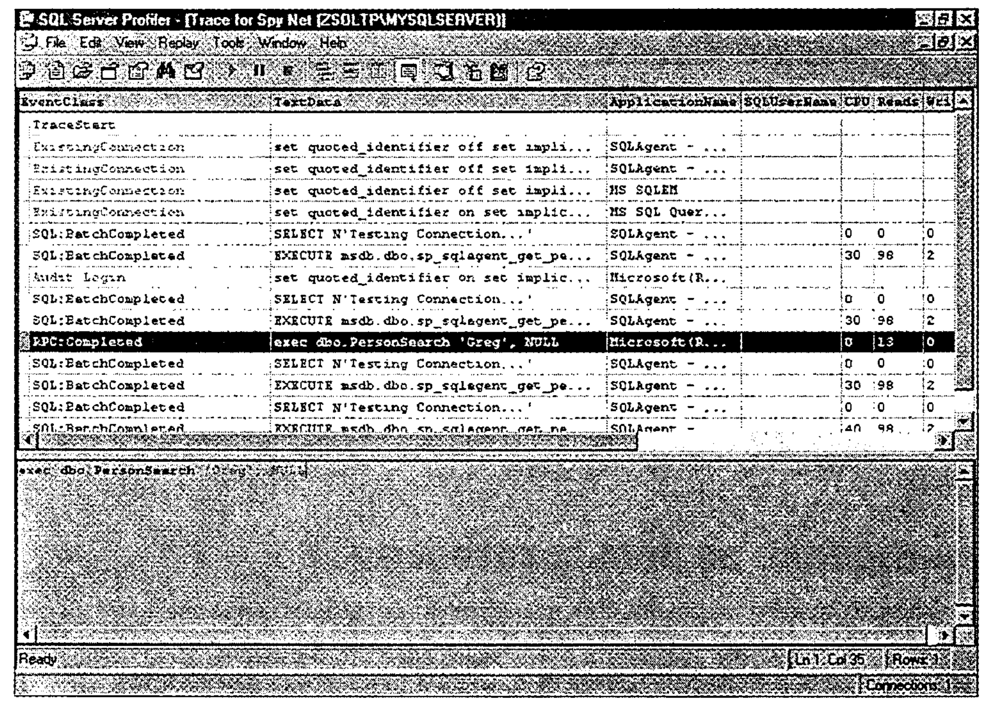


Рис. 14.6. Слежка за пользователями; смотрим, что они делают

Отладка соединения с помощью средства *Client Network*

В главе 2, "Компоненты SQL Server 2000", кратко описаны некоторые инструменты SQL Server 2000, в том числе средство Client Network. С его помощью можно изменять способ подключения компьютера к SQL Server 2000.

В нашей установке SQL Server 2000 фактически доступен только протокол TCP/IP, однако на компьютерах с Windows NT/2000 доступны также другие протоколы. Помимо TCP/IP, наиболее распространенный — Named Pipes. Более подробную информацию об этих протоколах можно найти в приложении Б, "Установка и настройка SQL Server 2000".

Иногда возникают ошибки, которые кажется невозможным объяснить. Например, Web-узел и клиентское приложение работают нормально, тем не менее соединение не устанавливается.

Драйвер ODBC или другой механизм соединения сообщает, что с указанным сервером соединение не может быть установлено. Причин тому может быть несколько. Чаще всего решить эту проблему с соединением помогает замена TCP/IP протоколом Named Pipes.

Определить соединение Named Pipes можно с помощью либо средства Client Network, либо диспетчера ODBC (пиктограмма для его запуска находится в окне панели управления операционной системы). Поэтому, если возникли проблемы с соединением, попробуйте сначала изменить протокол.

Эта книга написана для SQL Server 2000 под управлением Windows 98, поэтому мы не сможем изменить протокол сети. Однако все же кратко рассмотрим эту процедуру для систем Windows NT/2000. Надо же что-нибудь предложить и пользователям NT/2000!



Не изменяйте эти опции, если нет проблем с соединением! Например, пытаясь изменить протокол на Named Pipes, вы можете обнаружить, что никакой протокол установить уже невозможно.

Средство Client Network можно запустить из группы программ SQL Server главного меню. Для этого выберите команду Start⇨Programs⇨Microsoft SQL Server⇨Client Network Utility. Появится окно, показанное на рис. 14.7.

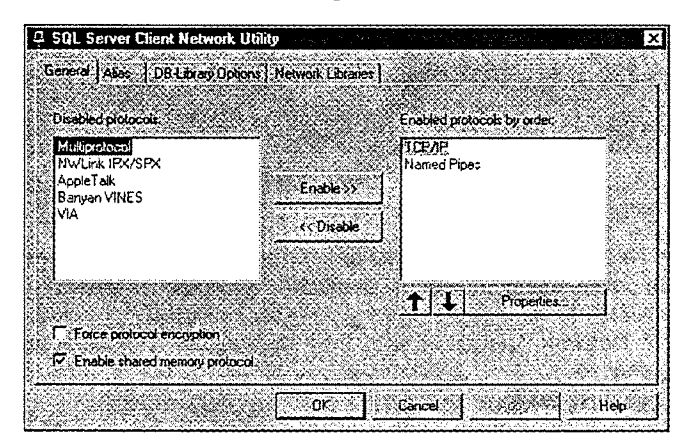


Рис. 14.7. Средство Client Network

Во вкладке General собраны протоколы, доступные для вашего типа установки. Сейчас я перешел на Windows 2000, поэтому на рис. 14.7 вы видите два подключенных протокола: TCP/IP и Named Pipes. В случае Windows 98 в этом поле вы увидите только один протокол — TCP/IP.

Активизируйте вкладку Alias (Псевдоним), как показано на рис. 14.8.

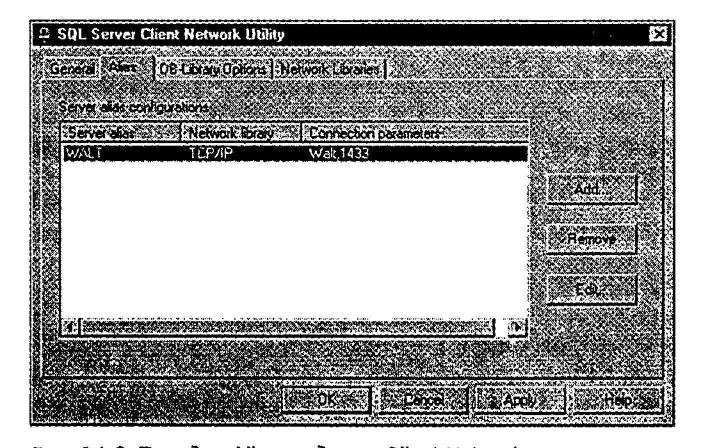


Рис. 14.8. Вкладка Alias средства Client Network

С помощью параметров этой вкладки можно настраивать способ общения с SQL Server. Здесь можно изменить используемый протокол. Для этого нужно в списке выбрать псевдоним (в данном случае WALT) и щелкнуть на кнопке Edit. Появится окно, показанное на рис. 14.9.

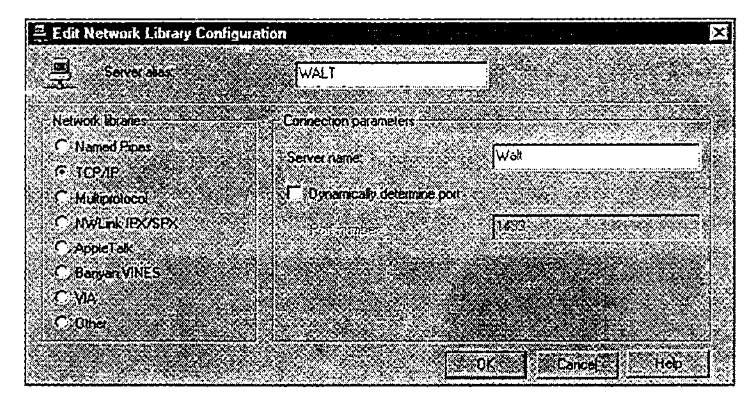


Рис. 14.9. Окно Edit Network Library Configuration (Настройка параметров сетевой библиотеки)

Если в группе Net libraries установить переключатель Named Pipes, то все клиентские приложения типа Query Analyzer, Enterprise Manager и им подобные будут общаться с SQL Server 2000 на основе протокола Named Pipes.

Средство Client Network поддерживает еще несколько параметров настройки, но они нам не понадобятся.

При работе с Client Network необходимо помнить следующее:

- если установлена операционная система Windows 98, то изменить протокол на Named Pipes нельзя;
- не следует менять протокол, если в этом нет крайней необходимости.

Отладка хранимых процедур

Исходя из своего опыта, могу с уверенностью утверждать: отладка хранимых процедур — одна из самых сложных задач. Однако теперь это уже не так! Реализация средства Query Analyzer в SQL Server 2000 позволяет отлаживать хранимую процедуру так, будто это программа в интегрированной среде разработки. Мы можем устанавливать точки прерывания, осуществлять пошаговое выполнение процедуры и т.д. Те, кому приходилось работать с хранимыми процедурами, оценят эти нововведения по достоинству.

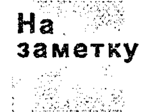
В предыдущих версиях (SQL Server 7.0 и Visual InterDev) тоже можно было делать это, однако сложность работы в режиме отладки сводила на нет все преимущества. Теперь же, используя новые инструменты отладки, достаточно в любом месте кода щелкнуть правой кнопкой мыши и выбрать команду Debug (Отладка) из появившегося контекстного меню.

Средство Query Analyzer в SQL Server 2000 предоставляет стандартные окна отладки. В окне Watch (Окно наблюдения) можно увидеть текущие значения переменных. В окне Callstack (Вызов стека) можно наблюдать вызванные, но не завершенные процедуры. Упростился переход из среды разработки в среду SQL Server 2000. Это особенно понравится разработчикам Access.

Вид нового отладочного окна в Query Analyzer показан на рис. 14.10.

```
CREATE PROCEDURE PersonSpyInsert
        @Firstname VARCHAR(50),
        @Surname VARCHAR (50),
        @DOB DATETIME = NULL,
        @SpyNumber VARCHAR(10) = NULL,
        @Alias VARCHAR(25) = NULL,
        @DateCommencedWork DATETIME,
        @AnnualSalary MONEY,
        @IsActive BIT = 1
AS
        INSERT INTO Person (Firstname, Surname, DOB)
        VALUES (@Firstname, @Surname, @DOB)
```

Рис. 14.10. Новое отладочное окно в Query Analyzer



Сегодня это средство доступно только для хранимых процедур, но я уверен, что в ближайшем будущем Microsoft значительно расширит его, охватив отладку представлений, функций, скомпилированных запросов и т.д. Правда, это всего лишь мое мнение; пока что, насколько мне известно, это еще не стало реальностью.

Вам мало места?

Одним из источников неполадок может стать нехватка наиболее ценных ресурсов компьютера — памяти и дискового пространства.

Если на сервере есть несколько баз данных, для их нормальной работы может не хватить дискового пространства. Когда это происходит, все базы данных выходят из строя.

Конечно, в нашем вымышленном примере SQLSpyNet такое может произойти, только если начнется третья мировая война! Однако в реальном мире нехватка места довольно часто вызывает серьезные проблемы, особенно в базах данных систем экстренного реагирования. В этом разделе рассматриваются возможные причины и способы предотвращения нехватки ресурсов, включая управление размерами файлов и журналов.

Как память влияет на транзакции базы данных

Если для базы данных оказывается недостаточно оперативной памяти, то возникают проблемы и частые сбои. Это объясняется тем, что большинство изменений дан-

ных, прежде чем они будут зафиксированы на диске, выполняются во временной базе данных tempdb. Если оперативной памяти достаточно, SQL Server 2000 размещает в ней как можно большую часть базы данных. Ведь чтение из оперативной памяти выполняется намного быстрее, чем с диска. Если же оперативной или дисковой памяти недостаточно, приходится использовать средства управления этими ресурсами.

В Enterprise Manager существует возможность просмотра объема выделенной и используемой памяти для файлов базы данных. Для просмотра этой информации щелкните на базе данных SQLSpyNet (или другой); появится окно, показанное на рис. 14.11.

Рис. 14.11. Выделенная и используемая память для базы данных SQLSpyNet



Если в окне *Enterprise Manager* не отображаются данные о памяти, выберите команду *View⇨TaskPad* (рис. 14.12).

Теперь рассмотрим, как сократить файлы данных для экономии дискового пространства.

Уменьшение размера базы данных путем сокращения файлов данных

Сокращение файлов данных не означает их упаковки, подобно тому как это делают программы архивирования.

При сокращении файлов данных удаляются неиспользуемые страницы данных. Например, если таблица имела пять страниц, в которых хранились данные, и мы

удалили данные из двух страниц, то SQL Server 2000 будет по-прежнему полагать, что все пять страниц предназначены для данных этой таблицы.

Сокращая файл данных, мы избавляемся от этих двух страниц, которые удерживались таблицей. При этом, однако, возникают некоторые сложности.

Рис. 14.12. Изменение вида отображения окна Enterprise Manager



Нельзя сократить *всю* базу данных до размеров, меньших, чем она имела при создании. Например, наша база данных SQLSpyNet создана размером 5 Мбайт. Если она вырастет до 25 Мбайт, то мы не сможем сократить ее до 3 Мбайт.

Что делать, если файлы данных стали слишком большими? В отличие от *всей* базы данных, отдельный файл можно сократить до размеров, меньших, чем ему было выделено при создании. Каждый файл данных нужно сокращать отдельно, используя для этого оператор Transact-SQL `DBCC SHRINKFILE`, который повторно выделяет для указанного файла данных дисковое пространство.



Для выполнения оператора Transact-SQL `DBCC SHRINKFILE` нужно открыть окно *Query Analyzer* и выделить базу данных SQLSpyNet.

Изменение размера файла журнала транзакций

Сокращение журнала транзакций несколько отличается от сокращения файла данных, что связано со временем выполнения этого процесса.

Когда сокращается файл данных, это выполняется практически немедленно. Для журнала транзакций это не так. Сначала файл SQL Server 2000 уведомляется о том, что журнал транзакций подлежит сокращению. Теперь SQL Server 2000 попытается сокращать журнал до указанных размеров, когда удаляются записи журнала или создается его резервная копия. Но если *активные* разделы находятся в конце журнала, то они не будут сокращены, пока не будут передвинуты в начало журнала.



Активный раздел журнала не может быть удален. Он используется для восстановления состояния базы данных в любой момент времени. Поэтому незавершенные транзакции должны быть сохранены, чтобы их можно было идентифицировать при отмене транзакции. Они всегда находятся в базе данных, чтобы в случае краха сервера ее можно было восстановить при его повторном запуске.

Активные транзакции, которые еще не зафиксированы и не отменены, помечаются как незавершенные. Более подробное толкование этих терминов приведено в главе 8, "Защита данных с помощью транзакций, блокировок и механизма обработки ошибок".

Иногда администраторы баз данных передвигают активные разделы журнала транзакций в начало журнала путем его заполнения фиктивными транзакциями. Конечно, этого нельзя сделать при "напряженной" работе базы данных.

Что определяет активный раздел журнала транзакций? Журналы транзакций состоят из меньших журналов, которые называются *виртуальными*.

Термин

Журнал транзакций состоит из *виртуальных журналов*. В них отмечено, какие транзакции активны. Когда активная транзакция фиксируется или отменяется, виртуальные журналы освобождаются для записи новых транзакций. Запись в начало файла журнала транзакций начнется только после достижения конца файла. Однако, если журнал окончательно заполнен (активными транзакциями), его размер увеличивается.

Например, если файл журнала транзакций нашей базы данных имеет размер 200 Мбайт, то он может состоять из пяти виртуальных журналов по 40 Мбайт каждый.

На заметку

Размер журнала транзакций и файлов данных устанавливается при первом создании базы данных (глава 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet"). Однако размеры журналов можно изменить в любой момент с помощью средства *Enterprise Manager* или путем изменения размера файла журнала в окне свойств базы данных, как описано в главе 3. Это можно сделать также, выполнив оператор Transact-SQL `ALTER DATABASE`. Более подробную информацию об этом операторе можно найти в справочной системе.

Если виртуальный журнал не содержит активных транзакций базы данных, он помечается как подлежащий сокращению. Занимаемая им память может быть повторно выделена для будущих транзакций.

Сокращение журнала транзакций

Сокращение журнала транзакций позволяет повторно разместить занимаемую им память, если больше нет активных транзакций. В базе данных SQLSpyNet установлено автоматическое выполнение этого процесса, однако в реальных системах так обычно не делают (глава 10, "Обеспечение доступности данных").

Мы сокращаем журнал нашей базы данных потому, что нам не нужен план воспроизведения в любой момент времени. В нашем примере более приемлемо восстановление ночной резервной копии, чем затраты на резервное копирование журнала транзакций каждые десять минут.

На заметку Во время сокращения файлов данных или журнала транзакций создать резервную копию базы данных или журнала транзакций нельзя. И наоборот, при выполнении резервного копирования нельзя сокращать указанные объекты.

На заметку Фактически журнал транзакций сокращается при каждом выполнении резервного копирования всей базы данных или журнала транзакций. SQL Server 2000 автоматически сокращает журнал транзакций. (Резервное копирование рассматривается в главе 10, "Обеспечение доступности данных".)

Объединение серверов в кластер

Когда ресурсы исчерпываются и управление памятью не помогает, можно попытаться добавить в систему новый сервер (или серверы), которые возьмут на себя часть нагрузки и предотвратят крах одиночного сервера. *Кластеризация* — это возможность создать группу серверов SQL Server, поддерживающих друг друга.

С помощью кластеризации можно наращивать размеры приложений до бесконечности, так как при нехватке ресурсов можно просто добавить еще один сервер. Разделение таблиц и распределенных представлений (см. главу 15, "Самостоятельное исследование SQL Server 2000") позволяет разнести их по разным серверам, но при этом обращаться с ними так, будто это одна таблица или представление.

В кластере можно иметь много узлов (т.е. серверов), но что произойдет, если один из узлов отказывает? На помощь приходит система поддержки отказоустойчивости кластеров. Так называется процесс, который определяет обработку краха одного их серверов кластера. Это напоминает перенос тяжелого предмета несколькими грузчиками. Если один из них споткнется, другие подхватят груз с его стороны. Поддержка баз данных становится при этом не сложнее прогулки в парке!

Кластеризацию в SQL Server 2000 легко установить и поддерживать, к тому же по сравнению с SQL Server 7.0 она значительно усовершенствована. Конфигурация кластера осуществляется при установке SQL Server 2000; для этого достаточно выбрать в мастере установки тип Virtual Server. Однако для полноценной работы кластера должна быть установлена программа MSCS (Microsoft Cluster Service). Описание процесса установки приведено в приложении Б, "Установка и настройка SQL Server 2000".

Таким образом, с помощью кластеризации можно достичь высокой отказоустойчивости приложений. Если сервер терпит крах, на его место становится другой сервер. Время выполнения запроса при этом увеличивается, но все базы данных остаются полностью работоспособными!

Резюме

Еще одна глава завершена, и мы сделали очередной шаг в познании SQL Server 2000 и поднялись еще на одну ступень мастерства в разработке и поддержке приложений баз данных.

В этой главе продолжалось описание инструментов SQL Server 2000, но под иным углом зрения: рассматривались инструменты, предназначенные для решения проблем (таких, например, как ошибки), возникающих при разработке приложения.

Возможности инструментов отладки приложений и простота их использования делают SQL Server 2000 прекрасным средством разработки приложений баз данных!

Следующие шаги

В следующей главе рассматриваются некоторые совершенно новые средства SQL Server 2000. Вы также узнаете, как использовать справочные материалы Books Online — неиссякаемый источник информации для разработчика и администратора баз данных SQL Server.

ГЛАВА 15

Самостоятельное исследование SQL Server 2000

В этой главе...

Работу над этой главой я предвкушал с особым нетерпением! Ведь здесь можно беспрепятственно испробовать совершенно новые замечательные средства SQL Server 2000. Кроме того, здесь я покажу вам вещи, которые, хотя и не имеют непосредственного отношения к нашему приложению, но очень пригодятся при разработке и поддержке приложений SQL Server.

Изменения в SQL Server 2000 не были столь радикальными, как в SQL Server 7.0 по сравнению с SQL Server 6.5, и все же появилось довольно много новых средств.

В этой главе рассматриваются некоторые мастера, облегчающие разработку приложений, и способы работы с многими экземплярами SQL Server 2000, как и с многими серверами с одной базой данных.

Что бы вы ни исследовали в SQL Server 2000, вам понадобится дополнительная информация. Здесь рассматриваются способы работы со справочными материалами Books Online и документация по DTS, предоставленная в SQL Server 2000 (в которой вы увидите, как решить небольшую проблему при работе с DTS, поставленную перед вами в главе 13, "Сбор разрозненных данных в один источник").

Использование справочных ресурсов при исследовании SQL Server 2000

Какая документация есть по SQL Server 2000? Намного легче было бы ответить на вопрос, какой документации *нет* по SQL Server 2000.

Компания Microsoft позаботилась, чтобы любая документация была доступна в любой момент, когда в ней возникнет необходимость.

Материалы Books Online поставляются с SQL Server 2000, в том числе материалы по более ранним версиям (где это необходимо) и по родственным технологиям, таким

как ADO и SQL DMO. Эти всеобъемлющие материалы помогут узнать все, что вам потребуется при работе с SQL Server 2000.

Например, если вы не вполне уверены в правильности написания оператора CREATE TABLE, то можете обратиться к Books Online. Если вам нужен обзор средств анализа данных, в Books Online вы можете получить о нем любую информацию.

Как же запустить справочную систему Books Online? Существует два способа.

- С помощью команды Start⇨Programs⇨Microsoft SQL Server⇨Books Online (рис. 15.1).

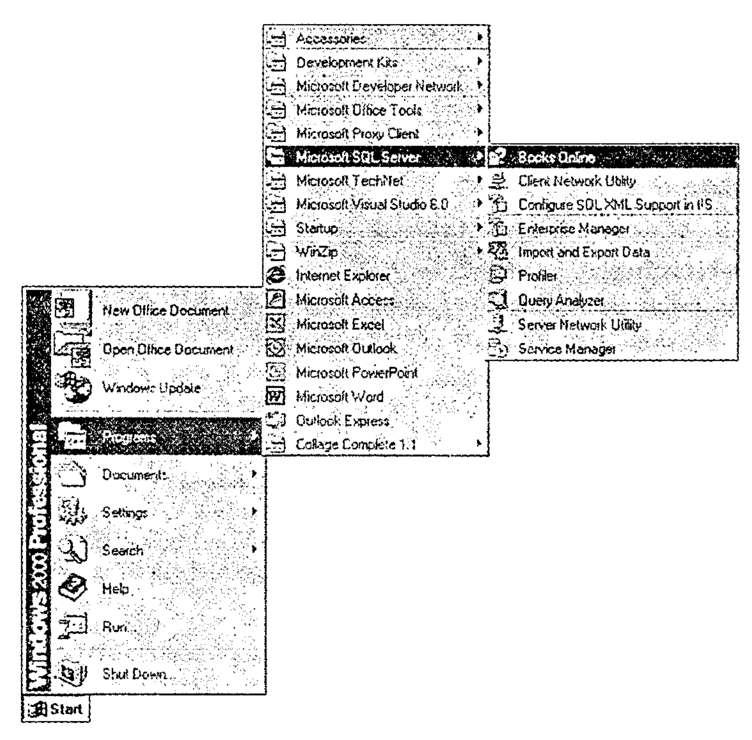


Рис. 15.1. Доступ к Books Online посредством команд меню Start

- С помощью клавиши <F1> в Enterprise Manager, Query Analyzer или в любой другой утилите SQL Server 2000. Обычно эта подсказка контекстно-зависима. Если в Enterprise Manager при открытом окне дизайнера таблиц нажать клавишу <F1>, то открывается окно Books Online, содержащее информацию о дизайнере таблиц.

Поиск и чтение информации в справочной системе *Books Online*

Как получить нужную информацию в Books Online? Существует три основных способа поиска информации. Скорее всего, вы уже знакомы с вкладками Contents (Содержание), Index (Указатель) и Search (Поиск). В этом разделе я дам вам ряд советов, которые помогут ускорить поиск нужной информации.

В справочной системе Books Online реализован механизм поиска, подобный механизму поиска в Internet. Например, если нужны сведения о средстве анализа данных (Analysis Services), то система вернет намного больше информации при вводе

Analysis Services, а не **"Analysis Services"** (с двойными кавычками). Это объясняется тем, что в первом случае возвращаются все темы, содержащие слова *Analysis* и *Services*, а во втором — только темы, содержащие фразу *Analysis Services*.



Используйте двойные кавычки продуманно, потому что они могут ограничить поиск значительно больше, чем вы предполагаете.

Как и в этой книге, в справочной системе Books Online для облегчения поиска используются легкоузнаваемые типы шрифтов.

- `Моноширинный` — для отображения кода программ, сообщений об ошибках и вывода информации на экран.
- `ПРОПИСНЫЕ СИМВОЛЫ` — ключевые слова Transact-SQL, например `CREATE PROCEDURE`.
- *`Моноширинный курсив`* — для указания того, где нужно вводить имена объектов, например `CREATE TABLE` *`имя_таблицы`*.
- **Полужирный** — для имен хранимых процедур, названий типов переменных, имен таблиц и т.д. Этот шрифт указывает на то, что слово должно быть набрано *точно* так же, как написано, например **sp_help**.

В справочной системе Books Online есть также ряд элементов, которые помогут найти нужную информацию.

- Глоссарий терминов (Glossary of terms). Содержит ключевые слова и их краткие описания.
- Расширенный текст (Expanding text). Отмечен знаком "плюс" (+) рядом с текстом. Если щелкнуть на нем, текст разворачивается и можно увидеть дальнейшие подсказки.
- Связанные темы (Related topics). Отмечены пиктограммой в верхней правой части темы. Если щелкнуть на пиктограмме, то отображаются связанные темы.
- Гиперссылки (Hyperlinks). Средство перемещения по связанным темам в справочной системе Books Online или на Web-узле, предоставляющем справку по данной теме. Отмечены цветным подчеркнутым текстом.
- Основная страница (Home page). Справочная система SQL Server 2000 Books Online создана в виде гипертекста и работает по принципу броузера. В ней есть основные средства навигации броузера, включая кнопки Home, Forward, Back, Previous, Next и т.д.
- Избранное (Favorites). Папка Favorites аналогична такой же папке в Internet Explorer. Если вам часто требуется некоторая тема, можете добавить ее в папку избранных. Для этого нужно щелкнуть на вкладке Favorites, а затем на кнопке Add (рис. 15.2).

Books Online — огромная система; на первый взгляд она может показаться крайне сложной, однако она содержит довольно простое средство получения необходимой информации — контекстно-зависимую справку.

Допустим, вы импортируете данные из электронных таблиц Excel с помощью мастера импорта/экспорта. Если при этом щелкнуть на кнопке Help или нажать клавишу <F1>, то справочная система немедленно предоставит вам темы, касающиеся импорта данных посредством DTS.

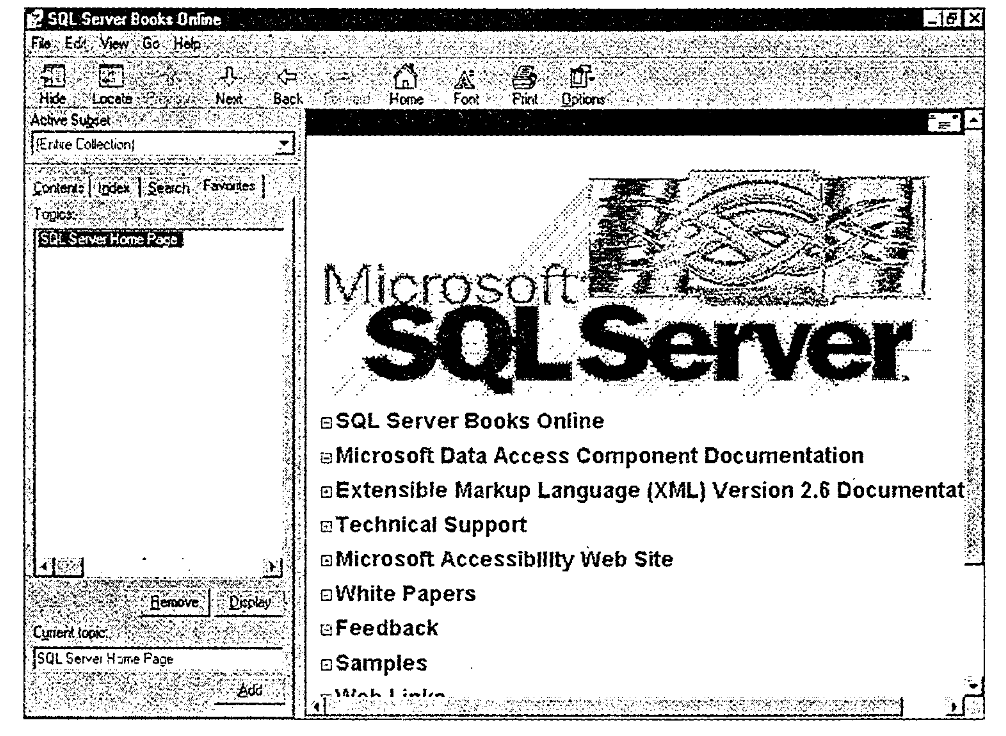


Рис. 15.2. Добавление темы в папку избранных тем справочной системы Books Online

Поиск справочной информации в Web

Иногда возникают проблемы, для решения которых информации в справочной системе Books Online оказывается недостаточно. Компания Microsoft учитывает это и предоставляет разработчикам и администраторам баз данных другие источники документации.

- MSDN (Microsoft Developer Network). Это почти неисчерпаемый источник информации для разработчиков. Комплект MSDN распространяется по подписке, однако большую часть информации можно найти по адресу: `http://msdn.microsoft.com` (причем бесплатно!). Набор MSDN содержит примеры кода по многим темам, включая Visual Basic, VBScript, Visual C++, SQL Server, ADO. Назовите тему, и вы увидите — она там есть!
- MSTN (Microsoft TechNet). Это в большей степени источник технической информации. В нем можно найти сведения о "жучках" различных систем и о способах решения распространенных проблем. Как и MSDN, он распространяется по подписке, однако доступен также в Web по адресу: `http://www.microsoft.com/technet`, причем тоже бесплатно. Здесь вы найдете ответы на наиболее распространенные вопросы.
- Web-узел SQL Server компании Microsoft. Содержит массу полезной информации, включая выдержки из книг для разработчиков, профессиональные советы, лучшие разработки, интерактивные конференции и т.д. Его адрес — `http://www.microsoft.com/sql`.

- PASS (Professional Association for SQL Server). Этот узел содержит много полезных советов администраторов баз данных, разработчиков и т.д. Это некоммерческая организация, стремящаяся содействовать внедрению и распространению SQL Server. Адрес узла — `http://www.sqlpass.org`.
- Существуют также многочисленные группы новостей, в которых регулярно обсуждаются темы, связанные с SQL Server.

Помните: все эти группы, интерактивные источники и книги созданы для того, чтобы помочь *вам*. Воспользуйтесь этой помощью, иначе их усилия пропадут даром!

Если вы обращаетесь к справочным материалам Books Online, значит, вы неплохой администратор баз данных. Нет ничего хуже, чем новичок, не перестающий докучать более опытным коллегам, потому что он не знает, где найти ответы на простейшие вопросы. Поэтому сначала исследуйте, а потом спрашивайте.

Однако если какая-либо проблема оказалась слишком трудной и борьба с нею затягивается — зовите на помощь! Большинство администраторов баз данных с удовольствием оторвутся от своей работы, чтобы помочь вам. В конце концов, они ведь знают, как трудно новичку. Как говорили древние: "Лучше выглядеть дураком несколько минут, чем остаться им на всю жизнь".

Мы кратко рассмотрели, как получить информацию по интересующей вас теме. Теперь рассмотрим "конфетки" SQL Server 2000, которые можно применить в нашем приложении.

Двойные, тройные... кто больше?

Наступит момент, когда вы обнаружите, что запускаете несколько экземпляров SQL Server (разных версий) на разных компьютерах и с данными, расположенными в разных местах. Однако сейчас, чтобы все было просто и синхронизировано, рассмотрим возможности некоторых новых средств SQL Server 2000.

Запуск многих экземпляров на одном компьютере

SQL Server 2000 позволяет запустить на одном компьютере несколько экземпляров сервера одновременно. В прошлом можно было запустить на компьютере только один экземпляр SQL Server, который в SQL Server 2000 называется экземпляром по умолчанию. Сейчас их может быть много. Компания Microsoft протестировала запуск 15 именованных экземпляров и одного экземпляра по умолчанию! Мне кажется, это больше, чем кому-либо когда-нибудь может понадобиться.

Зачем это нужно? В прошлом, когда клиентские базы данных были очень маленькими, а их параметры безопасности устанавливались отдельно, часто требовалось несколько серверов, чтобы настраивать непосредственно каждую базу данных. Это приводило к серьезным затратам для поставщиков программного обеспечения.

Когда поддерживается несколько экземпляров SQL Server 2000, каждый из них совершенно отделен от других, фактически это то же, что и несколько серверов. Каждый из них можно конфигурировать отдельно, совершено независимо от других.

Экземпляры имеют только два общих элемента: инструменты клиентов и компоненты MDAC (Microsoft Data Access Components). Сюда входят Query Analyzer, Enterprise Manager и SQL Profiler. Поэтому, устанавливая новые версии MDAC, нужно соблюдать осторожность, поскольку они могут повлиять на каждый экземпляр сервера.

Как удалось компании Microsoft достичь этого? В реестре (рис. 15.3) SQL Server 2000 создает новую запись для каждого экземпляра, а в локальной файловой системе все необходимые файлы помещаются в папку MSSQL$*имя_экземпляра*, что позволяет использовать для каждого экземпляра отдельную папку.

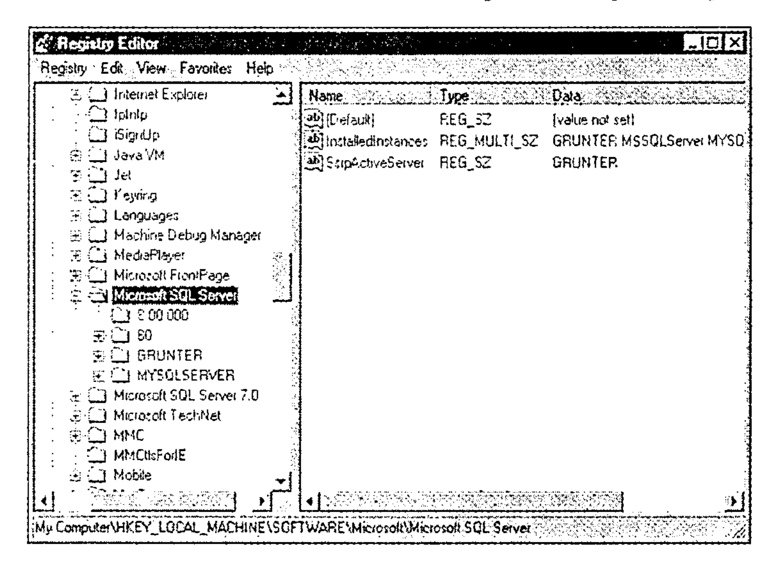


Рис. 15.3. Конфигурация реестра с новым экземпляром SQL Server 2000



Чего следует остерегаться? Каждый экземпляр — это полностью работоспособная копия SQL Server 2000, поэтому, если установить несколько экземпляров, можно быстро исчерпать дисковое пространство.

И еще. Каждый экземпляр SQL Server 2000 использует всю доступную оперативную память компьютера. Однако по умолчанию память может использоваться совместно с другими приложениями. Если, например, компьютер имеет 128 Мбайт оперативной памяти и SQL Server 2000 забрал всю доступную свободную память, то он освободит ее, когда она понадобится другим приложениям.

Каждый экземпляр SQL Server 2000 интенсивно использует ресурсы системы, поэтому Microsoft не рекомендует устанавливать слишком много экземпляров на одном компьютере. Если в ваших базах данных интенсивно выполняются громоздкие транзакции, разместите их на отдельных компьютерах. Тогда ресурсы не будут разделяться, и эффективность выполнения будет выше.

Поддержка многих экземпляров — это совершенно новое средство, только сейчас предоставленное разработчикам, которым необходимо переносить приложения после тестирования в рабочую среду. Это средство особенно полезно при разработке приложений, требующих различных параметров сервера.

Использование разных способов сортировки

В предыдущих версиях SQL Server для всех баз данных этого сервера применялись только установленные параметры сортировки.

На заметку Настройка параметров сортировки рассматривается в приложении Б, "Установка и настройка SQL Server 2000".

Однако теперь ситуация улучшилась. В SQL Server 2000 можно задать параметры сортировки не только для всего сервера, но и для каждой базы данных. Более того, параметры сортировки можно задать не только для базы данных, но и для каждого столбца таблицы! Это позволяет более гибко манипулировать данными, полученными из разных стран. Можно установить поддержку расширенного набора символов для японских, арабских и практически любых символьных данных.

Поскольку доступны различные конфигурации сортировки, можно также устанавливать разные способы упорядочения, задавать чувствительность к регистру и т.д.

Поддержка согласованности многих баз данных при репликации

Репликация — мощное средство SQL Server 2000, позволяющее поддерживать согласованность многих баз данных. Это значит, что часть пользователей может обновлять данные в Сибири, а часть — в Данедине (Новая Зеландия).

Изменения, внесенные пользователями, могут быть реплицированы в другие базы данных. Таким образом, пользователи в Сибири увидят в своих базах данных все изменения, сделанные в Данедине, и наоборот. Такой тип репликации называется *двусторонней репликацией*. Данные переносятся с одного сервера на другой в обоих направлениях.

При *односторонней репликации* локальные базы данных передают свои данные на центральную базу данных, которая принимает их, но не отправляет обратно.

В настоящее время существует CE-версия SQL Server 2000, работающая под управлением операционной системы Windows CE на переносных компьютерах. Это позволяет вводить данные в компьютеры с Windows CE в "походных условиях", сохранять эти данные, а затем, вернувшись в офис, реплицировать их на центральный сервер. Какая гибкость!

Однако новейшая инициатива Microsoft — платформа .NET — позволяет достичь еще большей гибкости. С ее помощью можно создавать собственные интерфейсы для Windows CE с помощью языков Visual Basic, C#, C++ и т.д. Таким образом, интерфейс, хорошо знакомый и привычный для пользователя, можно преподнести ему на "блюдечке с голубой каемкой"! Компактный компьютер, например Compaq iPAQ, может даже иметь красивый цветной экран. Это — будущее компьютерной техники!

Использование расширенных возможностей разработки

Новые средства SQL Server 2000 позволяют упростить разработку и улучшить целостность данных. В этом разделе рассматриваются некоторые приемы работы со средством Query Analyzer и другими инструментами разработки.

Использование каскадной декларативной ссылочной целостности

Создавая базу данных, мы с помощью графических инструментов задавали определенные отношения между таблицами, а создавая эти отношения, видели параметры реализации каскадной декларативной ссылочной целостности (рис. 15.4). Тогда мы не воспользовались ими. И напрасно. Это новое средство SQL Server 2000 особенно оценят те, кому в прошлом для поддержки этого типа целостности приходилось вручную создавать громоздкие триггеры.

Термин *Декларативная ссылочная целостность* данных обеспечивается ограничениями внешнего ключа, реализованными в отношениях между таблицами, и гарантирует, что при обновлении данных оператором UPDATE их целостность не будет нарушена. Этот тип целостности также можно реализовать путем определения для таблиц триггеров, пользовательских значений по умолчанию и т.д. Способы реализации декларативной ссылочной целостности рассматривались в главе 3, "Вербовка "виртуальных" агентов, или Создание базы данных SQLSpyNet".

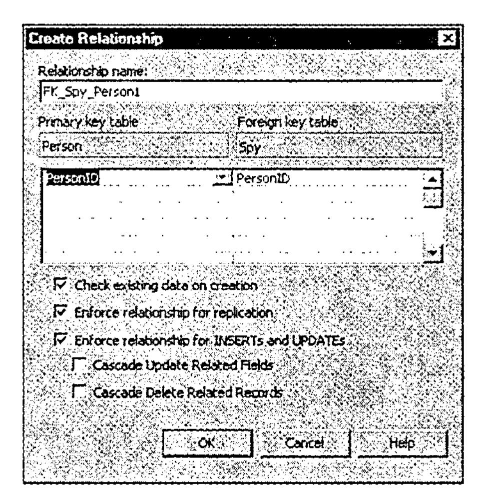


Рис. 15.4. Декларативная ссылочная целостность в SQL Server 2000

Если установлена каскадная декларативная ссылочная целостность, то при удалении родительских записей автоматически удаляются соответствующие дочерние записи связанной таблицы. В реляционных базах данных родительская запись удаляется только после удаления всех связанных с нею дочерних записей. В прошлом для поддержки этих условий приходилось создавать сложные триггеры и хранимые процедуры. Теперь SQL Server 2000 делает это за нас!

Сказанное относится не только к удалению, но и к обновлению записей. Если первичный ключ столбца изменяется (упаси Бог!), то соответствующим образом изменяются все записи вторичных ключей. Это же изменение выполняется для вторичных ключей следующей связанной таблицы (если она есть). Таким образом изменение распространяется на всю базу данных.

Теперь программисты Access не будут иметь проблем при переносе данных в SQL Server 2000!

Имитация встроенных функций с помощью пользовательских функций

Пользовательские функции — это чудо! Создание одной уже описано в главе 6, "Использование функций для повышения эффективности управления информацией". Пользовательская функция представляет собой гибкий модуль Transact-SQL, возвращающий *скалярные значения* или *таблицы*.

Термин Возвращаемое *скалярное значение* — это одно значение, тип которого определен в операторе CREATE FUNCTION.

Термин Возвращаемая *таблица* — это таблица результата. Она представляет собой новый тип данных.

В пользовательские функции можно передавать один или несколько параметров. Тип параметров может быть любым, кроме нового типа данных TABLE, TIMESTAMP и CURSOR. Пользовательские функции можно применить в операторе SELECT, предложении WHERE и т.д. Они используются так же, как любые другие встроенные функции SQL Server 2000.

Пользовательскую функцию можно связать с таблицей, в этом случае структуру таблицы можно изменить, только удалив функцию. Внутри пользовательской функции можно ссылаться на встроенные функции SQL Server 2000. Это позволяет получить доступ к имени сервера, версии SQL Server и т.д. По моему мнению, это одно из лучших средств SQL Server 2000.

Программирование с тремя новыми типами данных

Версия SQL Server 2000 поддерживает три новых типа данных.

- SQL_VARIANT — специальный, позволяющий хранить данные любого типа, кроме SQL_VARIANT, NTEXT, TEXT и TIMESTAMP. Благодаря слабой типизации, он позволяет делать структуру данных более гибкой.
- BIGINT — 8-байтовое целое. Такой тип данных полезен в очень больших базах данных, содержащих более двух миллиардов строк. Таким образом, SQL Server 2000 сейчас поддерживает TINYINT, SMALLINT, INT и BIGINT.
- TABLE — самый интересный, позволяющий создавать временные таблицы как тип данных. В отличие от обычной временной таблицы, такая таблица не остается в контексте базы данных, поэтому, окончив работу с ней, не нужно заботиться о ее удалении.

Эти новые типы данных позволяют достичь большой гибкости при разработке и реализации приложений.

С помощью типа данных SQL_VARIANT можно делать структуру базы данных более гибкой. Тип BIGINT позволяет хранить в таблицах миллиарды строк, а с помощью типа TABLE можно создавать временные таблицы, предназначенные только для кратковременного использования и не затрагивающие всю базу данных.

Что еще можно делать с помощью *Query Analyzer*?

Окно Object Browser (рис. 15.5) в Query Analyzer — одно из наиболее примечательных нововведений в последней версии SQL Server 2000.

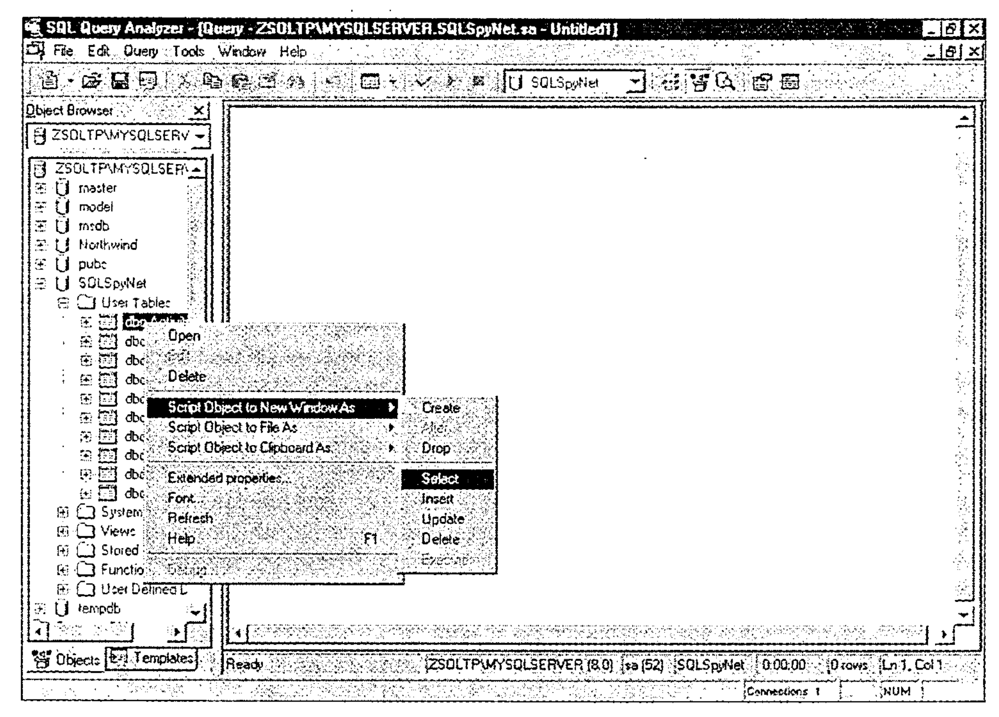


Рис. 15.5. Окно Object Browser в Query Analyzer

Это окно значительно облегчает передвижение по серверу. С его помощью можно получить доступ к базам данных, таблицам, представлениям, хранимым процедурам. Но это еще не все. С его помощью можно легко и быстро получить заготовки сценариев обработки объектов, т.е. удаления, создания, выбора объектов и т.д. Например, если щелкнуть правой кнопкой мыши на таблице и выбрать команду Script Object to New Window As⇨Select (Сценарий обработки объекта в новом окне⇨Выбор), то будет создана заготовка оператора SELECT. Останется только заполнить пробелы, и мы получим оператор выбора данных из таблицы. Это значит, что можно программировать "по образцу".

Окно Object Browser предназначено не только для пользовательских таблиц и хранимых процедур. В нем можно выбирать встроенные функции SQL Server 2000. В этом случае будет возвращен список параметров функции, их типы данных и тип возвращаемого функцией результата. Это отнимает значительно меньше времени, чем поиск, например, типов параметров функции в справочной системе Books Online.

С помощью Object Browser можно хранить шаблоны. Некоторые предопределенные шаблоны поставляются с SQL Server 2000, но можно создавать и собственные. Таким образом, можно создать общую структуру хранимых процедур, представлений и пр., которой будут придерживаться все разработчики проекта. Это внесет в разработку проекта некоторые элементы стандартизации.

С помощью окна Object Search (еще одно новое средство) можно ввести критерии поиска объекта (например, имя таблицы), и SQL Server 2000 выполнит в базах данных поиск объекта, удовлетворяющего заданным критериям. Это средство оказывается весьма полезным, например, когда требуется найти хранимую процедуру среди сотен других хранимых процедур базы данных (рис. 15.6).

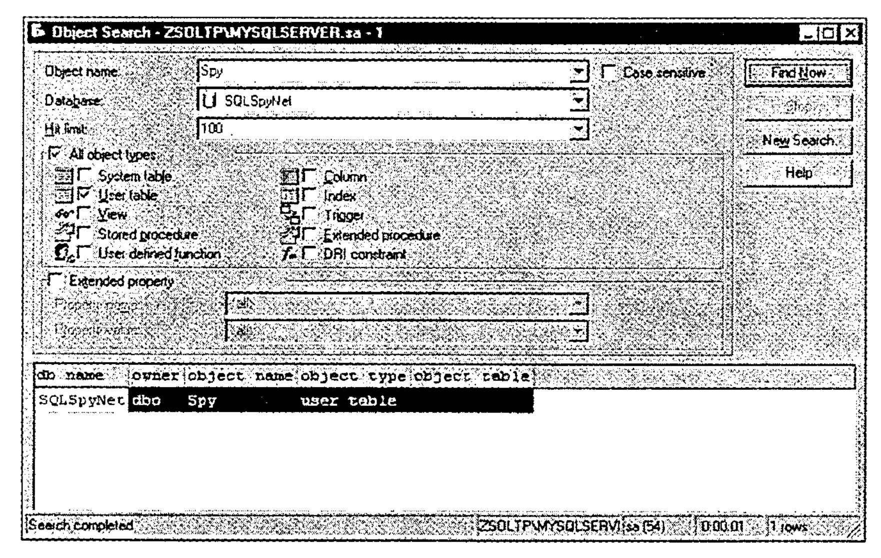


Рис. 15.6. Окно Object Search в Query Analyzer

Новая сетка данных в Query Analyzer позволяет возвращать многие результирующие наборы в одно окно. В предыдущих версиях результирующие наборы выводились каждый в свое окно. И хотя это нововведение не выглядит существенным, созданные им удобства весьма ощутимы.

Еще одна новинка. В окне Customize средства Query Analyzer можно создавать комбинации клавиш для быстрого вызова наиболее часто используемых операторов Transact-SQL. Например, можно установить, чтобы при нажатии <Alt+F1> выполнялась хранимая процедура sp_help. Чтобы открыть окно Customize (рис. 15.7), нужно выбрать команду Tools⇨Customize.

Сценарии объектов базы данных

Возможность генерировать сценарии объектов базы данных — еще одно замечательное нововведение SQL Server 2000. Допустим, есть два узла: разработки и рабочий. Когда в узле разработки реализуются какие-либо *структурные* изменения базы данных, они должны быть перенесены в рабочий узел. Для этого нельзя использовать мастер копирования базы данных, потому что он уничтожит в рабочем узле все данные. В то же время, сгенерировав сценарий измененного объекта, с его помощью можно перенести структурные изменения, не затрагивая данных в рабочем узле.

Сгенерируем сценарий одной из наших хранимых процедур. Запустите Enterprise Manager и откройте папку Stored Procedures. Найдите хранимую процедуру PersonSpyInsert. Щелкните на ней правой кнопкой мыши и выберите команду All Tasks⇨Generate SQL Script (рис. 15.8).

Появится окно (рис. 15.9), в котором можно задать различные опции создания сценария.

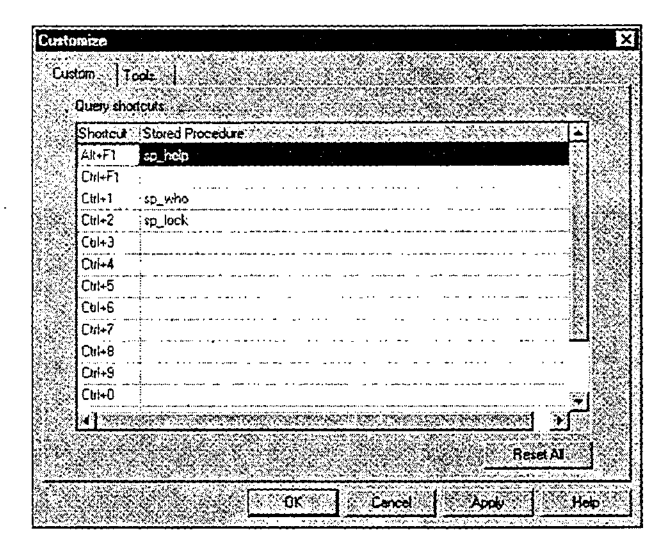


Рис. 15.7. Окно Customise в Query Analyzer

Рис. 15.8. Команда Generate SQL Script в Enterprise Manager

В этом окне фактически можно создать сценарий *всей* базы данных. В нем можно генерировать сценарии SQL для следующих элементов базы данных:

- таблиц;
- представлений;

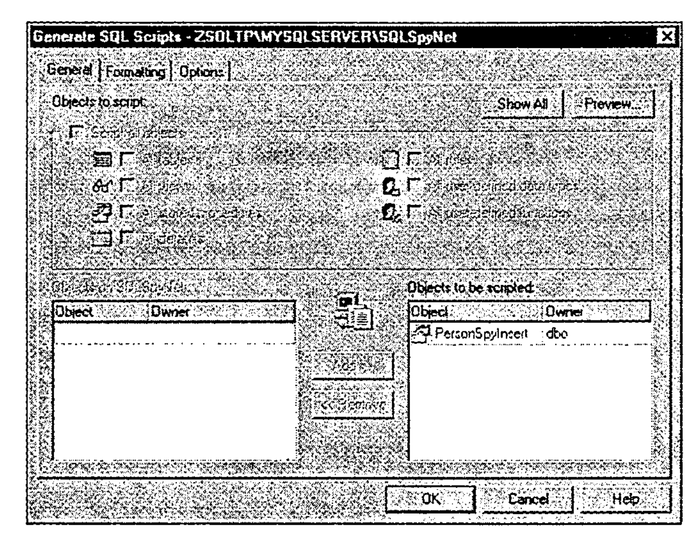


Рис. 15.9. Окно Generate SQL Script

- хранимых процедур;
- значений по умолчанию;
- бизнес-правил;
- пользовательских типов данных;
- пользовательских функций.

Во вкладке Formatting можно задать формат сценария. Например, можно указать, создавать ли оператор `DROP` для каждого объекта.

Во вкладке General можно предварительно просмотреть созданный сценарий, прежде чем сохранить его на диске. Это позволяет удостовериться, что сценарий создан так, как нужно.

Файл со сценарием сохраняется с расширением `.sql`. Этот файл можно посмотреть в обычном текстовом редакторе, потому что это всего лишь текстовый файл.

Генерация сценариев предоставляет большую гибкость при переносе изменений с тестового узла (или узла разработки) на рабочий. Уверен, это средство вам очень пригодится.

Использование триггеров AFTER и INSTEAD OF

О триггерах речь шла в главе 5, "Использование языка определения данных для просмотра и обновления информации". Сейчас мы вернемся к этому вопросу, но рассмотрим его в аспекте нововведений в SQL Server 2000.

Предыдущие версии SQL Server поддерживают триггеры уже очень давно. Однако в SQL Server 2000 введены некоторые расширения, касающиеся способов срабатывания и выполнения триггеров. Кроме того, значительно расширены средства управления триггерами.

SQL Server 2000 поддерживает два типа триггеров.

- `AFTER`. Эти триггеры срабатывают после заданного события. Например, для таблицы `Spy` в главе 5 мы определили триггер `AFTER`. Он срабатывает *после* вставки данных в таблицу. Этот тип триггеров нельзя создать для представле-

ний. Для одного события в таблице (INSERT, UPDATE или DELETE) можно определить много триггеров AFTER. Можно также задать последовательность их срабатывания.

- INSTEAD OF (BEFORE). Это новый тип триггеров в SQL Server 2000, которые перехватывают события перед их выполнением. Триггер INSERT, определенный для нашей таблицы Spy, перехватывает данные *перед* их вставкой в таблицу. В триггере можно обрабатывать эти данные, выполнять хранимые процедуры и т.д. Эти новые триггеры могут быть определены для таблиц.

Что это дает нам? С помощью этих новых расширений можно вводить бизнес-правила в SQL Server, не заставляя интерфейс пользователя (в двухуровневой модели) перехватывать их. Следовательно, если некоторое бизнес-правило изменяется, то понадобится вносить изменение только в одном месте.

Распределенные представления

С помощью этого нового средства можно *горизонтально разделить* таблицу и разместить ее на нескольких серверах.

Термин *Горизонтальное разделение* означает разбиение таблицы по горизонтали. Например, если у нас есть таблица, содержащая 10 000 строк, и 10 серверов, то мы можем поместить 1 000 строк на первый сервер, следующие 1 000 строк на второй и т.д.

Как это сделать? Разместим строки 1–1000 на первом сервере, 1001–2000 на втором и т.д. Для большей уверенности можно использовать параметр CHECK CONSTRAINTS, ограничивающий допустимые значения ключевого столбца.

Разделив таблицу, можно создать представление, которое свяжет все серверы. Таким образом, вся эта конструкция будет выглядеть как одна таблица.

Благодаря такой гибкости можно поддерживать огромную корпоративную среду или очень загруженный Web-узел. В будущем, обнаружив, что у нас недостаточно ресурсов, мы сможем просто добавить еще один сервер. Это делает возможности расширения воистину огромными!

Создание расширенных индексов

Индексы, как и триггеры, всегда были частью SQL Server, однако теперь их возможности значительно расширены.

Одно из самых больших нововведений — создание индексов для вычисляемого столбца. Вычисляемый столбец — это результат, например, суммирования. Рассмотрим столбец с именем SumOfAandB, вычисляемый как ColumnA+ColumnB. В прошлом невозможно было обратиться к этому столбцу быстро. Однако сейчас для этого столбца можно использовать индекс. Теперь SQL Server 2000 не должен вычислять значения столбца при каждом обращении к нему, он может извлечь результат из индекса. Чтение данных при этом значительно ускоряется.

Что еще изменилось в индексах? Сейчас можно определять индексы не только для вычисляемых столбцов, но и для представлений. Это делает представления очень "быстрыми". Однако SQL Server 2000 идет еще дальше. Если для представления определен индекс и вы считываете данные одной из его базовых таблиц, то SQL Server 2000 использует индекс представления при чтении данных из этой таблицы. Это значительно ускоряет выборку данных.

Можно задать порядок построения индекса (восходящий или нисходящий). При этом данные записываются на диск в указанном порядке, делая чтение более эффективным. Если некоторый столбец (например, Firstname таблицы Person) чаще всего считывается в восходящем порядке (т.е. имена выбираются по алфавиту), то для этого столбца следует определить восходящий индекс. Тогда при выборе каждой новой строки сервер сможет просто взять следующую в списке, а не искать ее заново. Скорость выполнения запроса при этом значительно увеличивается.

Создание индекса можно задать в базе данных tempDB. Это вынудит SQL Server 2000 перед созданием индекса упорядочить строки tempDB. Это несколько увеличивает требующееся при создании индекса дисковое пространство, однако скорость и эффективность построения индексов при этом возрастают, особенно если tempDB находится на другом диске.

Поддержка XML

Это новое расширение SQL Server 2000 обсуждается больше других. Довольно часто приходится слышать рекламу компаний, пытающихся вскочить на "подножку вагона" XML, однако Microsoft — несомненный лидер в этой области.

Как инструмент разработки реляционных баз данных, SQL Server 2000 поддерживает XML. Это значит, что результаты запросов можно получать в заданном формате XML. Если передать запрос к базе данных с помощью обычного оператора SELECT, в конце которого указано предложение FOR XML, то результат запроса будет возвращен в формате XML. Великолепно, не правда ли?

Рассмотрим следующий пример. В окне Query Analyzer выполните текст листинга 15.1.

Листинг 15.1. Выполнение оператора SELECT с выводом результата в формате XML

```
SELECT Firstname, Surname, DOB FROM Person
↳WHERE PersonID = 1 FOR XML AUTO
```

Результат запроса возвращается в формате XML (рис. 15.10).

Однако SQL Server 2000 не останавливается на этом. В системе Windows 2000 можно конфигурировать сервер IIS (Internet Information Server) таким образом, чтобы запросы к базе данных передавались непосредственно через адрес URL.

С помощью средства Configure SQL XML Support in IIS, поставляемого с SQL Server 2000, конфигурировать IIS довольно легко (рис. 15.11). Этот простой интерфейс разработан специально для настройки IIS.

Эти возможности делают вопрос безопасности данных особенно острым. Если запросы к базе данных можно передавать с броузера, значит, ваши данные открыты для всего мира. Однако все не так страшно. Существует возможность полного контроля запросов XPath. Можно также задать учетную запись, обязательную при выполнении запросов к базе данных.

Какие запросы может выполнять пользователь? В настоящее время в составе запроса XPath поддерживаются операторы SELECT и EXECUTE, однако с помощью расширения XML Updategrams можно использовать почти все операторы Transact-SQL, выполняемые в Query Analyzer, включая UPDATE, INSERT, DELETE, DROP и CREATE.

Зачем это может понадобиться? Это дает полный контроль над сервером, независимо от того, в какой точке земного шара вы находитесь.

Вы можете создать тщательно продуманную хранимую процедуру, выполняющую все ваши задачи администрирования. Затем эту хранимую процедуру можно вызвать с домашнего компьютера; таким образом, вы справитесь с задачей администрирования, не вставая с дивана!

SQL Query Analyzer - [Query - ZSOLTP\MYSQLSERVER.SQLSpyNet.sa - Untitled1*]

File Edit Query Tools Window Help

```
SELECT Firstname, Surname, DOB FROM Person WHERE PersonID = 1 FOR XML AUTO
```

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------------------
<Person Firstname="Richard" Surname="Darstidly" DOB="1965-07-15T00:00:00"/>

(1 row(s) affected)
```

Results

Query batch completed. ZSOLTP\MYSQLSERVER (8.0) sa (52) SQLSpyNet 0:00:00 1 rows Ln 1, Col 33

Connections: 1 NUM

Рис. 15.10. Новое предложение `FOR XML AUTO` *оператора* `SELECT` *в SQL Server 2000*

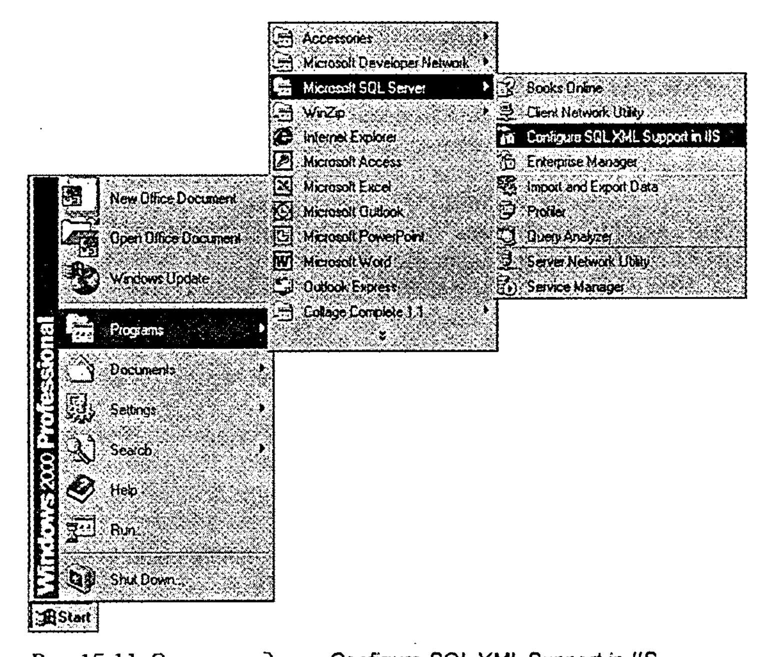


Рис. 15.11. Запуск средства Configure SQL XML Support in IIS

Использование новых мастеров SQL Server 2000

В системе SQL Server 2000 представлен мастер копирования базы данных и расширенная версия мастера настройки индексов. Рассмотрим эти мастера.

Настройка индексов

Мастер настройки индексов (Index Tuning Wizard) оценивает рабочую нагрузку сервера и дает рекомендации относительно создания оптимальных индексов и статистики базы данных. Этот мастер можно запустить из окна Enterprise Manager, выбрав команду Tools⇨Wizards. В диалоговом окне Select Wizard разверните ветвь Management, выделите Index Tuning Wizard (рис. 15.12) и щелкните на кнопке OK. Появится окно, показанное на рис. 15.13.

Термин *Рабочая нагрузка* — это сохраненный сценарий Transact-SQL или результат трассировки, полученный от средства SQL Profiler. Мастер проанализирует сценарий и даст рекомендации по повышению эффективности выполнения запросов.

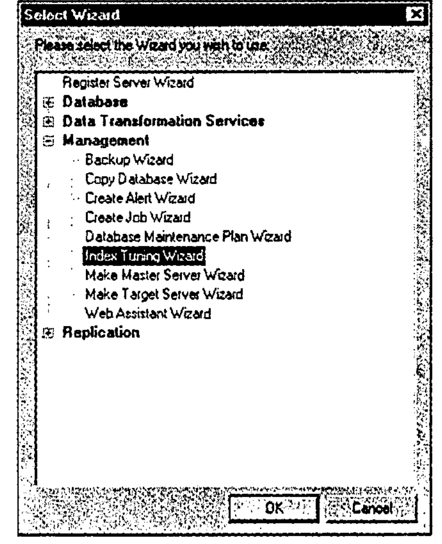


Рис. 15.12. Список мастеров в ветви Management

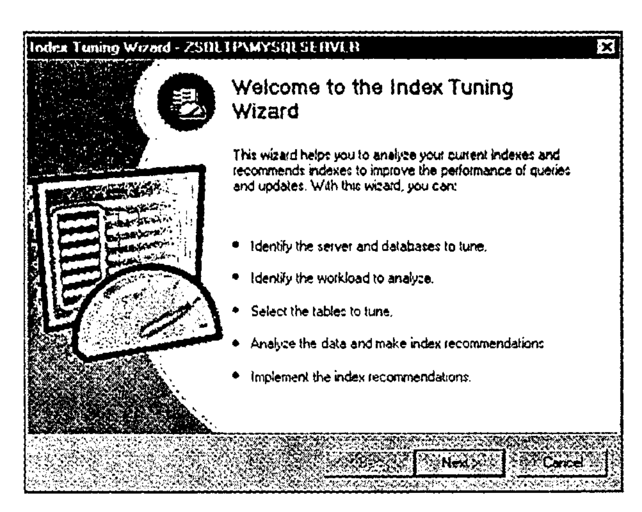


Рис. 15.13. Расширенный мастер настройки индексов в Enterprise Manager

Что дает использование мастера? Зачем же тогда нужен администратор базы данных? Вы совершенно правы! Однако с помощью мастера можно принять решение относительно того, как повысить эффективность, без трудоемкого анализа структуры данных и выполняемых задач. Мастер использует программу оптимизации запросов, которая анализирует запросы, представленные в рабочей нагрузке (статистике предыдущей работы сервера). На основе данных программы оптимизации мастер настройки индексов рекомендует состав и структуру индексов, оптимальные для данных запросов. Мастер оценивает влияние на производительность каждого индекса таблицы, представления или запроса и анализирует их эффективность при выполнении запросов, представленных в рабочей нагрузке.

Мастер также советует, что можно сделать для повышения эффективности запросов. С помощью расширенных опций вы можете изменить рекомендации мастера, приспособив их к потребностям решаемой задачи.

Копирование баз данных

С помощью мастера копирования баз данных можно переносить базы данных с одного сервера на другой или между экземплярами SQL Server 2000 на одном компьютере. С его помощью несложно также перенести базу данных с предыдущих версий (например, SQL Server 7.0) на SQL Server 2000. Все, что нужно, мастер сделает за вас! (Не все, конечно, однако вы меня поняли.)

Для запуска мастера копирования баз данных в Enterprise Manager выберите команду Tools⇨Wizards, разверните ветвь Management, выделите Copy Database Wizard (аналогично мастеру настройки индексов, см. рис. 15.12) и щелкните на кнопке OK. Появится окно, показанное на рис. 15.14.

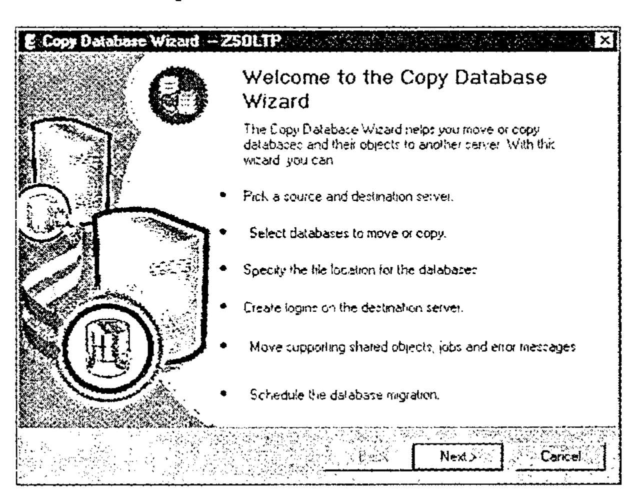


Рис. 15.14. Мастер копирования баз данных в SQL Server 2000

Вы, конечно, скажете: "Почему бы не сделать это с помощью процедуры резервного копирования и восстановления?" Действительно, так можно сделать. Однако, выполнив эту операцию таким образом, вы не перенесете все, что есть в базе данных, например учетные записи SQL Server 2000 или запланированные задачи.

Между тем мастер переносит всю базу данных полностью. Принцип его работы: "Прицелься и нажми на курок!" Можно даже установить время выполнения копиро-

вания, чтобы этот процесс протекал в период малой загрузки сервера. Это особенно полезно, если база данных очень большая.

Как вы понимаете, это чрезвычайно упрощает процесс переноса разработанной базы данных на рабочий сервер. Не нужны больше сложные сценарии считывания многочисленных зависимых объектов и последующей репликации их на другой сервер. Остается лишь простая, приятная операция.

Улучшение безопасности данных

При переходе в многопользовательскую среду управление безопасностью, учетными записями пользователей и ресурсами становится ключевым моментом обеспечения производительности и целостности приложения. Некоторые средства обеспечения безопасности поддерживаются только в Windows 2000.

Система Kerberos и делегирование прав доступа

К сожалению, мы не сможем воспользоваться этими новыми средствами, поскольку у нас не установлена Windows 2000 (возможно, это хороший повод для перехода на новую операционную систему).

Термин *Делегирование* — это способность передать задачу кому-нибудь другому. Это же относится и к SQL Server 2000. Устанавливая соединение со многими серверами, можно передать параметры пользователей с одного сервера на другой в том виде, как они созданы при первой регистрации пользователей.

Пусть, например, Джеймс зарегистрировался в нашем домене `SPYNET` (как `SPYNEY\James`) и установил соединение с первым экземпляром SQL Server 2000, который затем был соединен с другим сервером. В этом случае второй сервер будет знать информацию о соединении Джеймса, включая домен, откуда он пришел.

Как это работает? Система делегирования работает только под управлением Windows 2000 и при установленной на компьютере поддержке *Kerberos*. Серверы должны также использовать новые средства Active Directory системы Windows 2000. При этом должны быть установлены некоторые опции конфигурирования, которые можно найти в MSDN.

Термин *Kerberos* — это протокол безопасности, определенный документом RFC 1510 стандарта Internet. В протоколе Kerberos используются маркеры безопасности, определенные документом RFC 1964 стандарта Internet.

Какие еще расширенные средства безопасности есть в SQL Server 2000? Одно из новейших средств — применение паролей при копировании баз данных.

Использование паролей при резервном копировании баз данных

Этот вопрос кратко обсуждался в главе 10, "Обеспечение доступности данных". Защита резервного копирования с помощью пароля предотвращает возможность копирования и восстановления файлов резервной копии посторонними. Пароли

и опции резервного копирования *не* зашифрованы, однако они все же предоставляют некоторую защиту данных, которой не было в прошлом.

Резервная копия не может быть восстановлена без пароля, однако содержимое файлов резервной копии все же может быть скопировано.

Если возможные потери от несанкционированного копирования особенно велики, не полагайтесь только на защиту резервных копий паролем. В этом случае для обеспечения безопасности данных следует применять более действенные методы, среди которых можно назвать следующие:

- аутентификация пользователей с помощью Windows NT/2000;
- ограничение круга лиц, которым предоставлено право резервного копирования;
- физическое размещение сервера в закрытом помещении.

Использование аудита C2

Об аудите C2 речь шла главе 9, "Обеспечение безопасности базы данных Spy Net". SQL Server 2000 имеет аккредитацию C2 и полностью поддерживает аудит C2. Подробности можно найти на Web-узле Radium по адресу: `http://www.radium.ncsc.mil/tpep/epl/entries/TTAP-CSC-EPL-00-001.html`.

Аудит C2 — это набор правил безопасности, определенный Министерством обороны США. Это очень жесткие правила, они перехватывают практически все, что происходит в экземпляре SQL Server 2000.

Этот вопрос уже рассматривался ранее, сейчас лишь отметим следующее:

- аудит играет в SQL Server 2000 роль ключа: если результат проверки отрицательный, сервер останавливается;
- для перехвата событий проверки можно использовать средство SQL Profiler;
- все изменения параметров безопасности (GRANT, REVOKE, DENY, изменения паролей) включаются в проверку;
- аудит снижает производительность сервера.

Для достижения оптимальной производительности можно использовать более низкие уровни аудита. Дополнительные сведения по этому вопросу приведены в главе 9, "Обеспечение безопасности базы данных Spy Net".

Резюме

Что больше всего привлекает при написании книги? Возможность прикоснуться к новым замечательным средствам программирования. В этом отношении SQL Server 2000 меня не разочаровал.

Новые средства SQL Server 2000, основанные на гибкости и эффективности предыдущей версии SQL Server 7.0, предоставляют еще больше возможностей по созданию надежной и эффективной базы данных.

В настоящей книге рассмотрено довольно много новых средств SQL Server 2000, однако не подумайте, что этим они исчерпываются. Некоторые новые средства даже не упомянуты, например:

- полнотекстовый поиск;
- перенос журналов (хотя это кратко затронуто в обзоре репликации);

- расширения репликации;
- расширения средства преобразования данных.

И это только начало списка нерассмотренных новых возможностей.

Мне кажется, рассмотренные расширения на первых порах радикально повлияют на вашу повседневную работу. Со временем, когда вы лучше освоитесь с SQL Server 2000, вы будете использовать для решения более сложных задач другие новые средства, о которых я пока еще даже не подумал.

Мы рассмотрели также использование справочных материалов Books Online. Это неиссякаемый источник информации, который вы будете интенсивно использовать для дальнейшего расширения знаний. Материалы Books Online — один из лучших источников информации по SQL Server, однако характер их изложения предполагает, что вы уже понимаете основы компонентов SQL Server и принципы их работы.

Итак, дамы и господа, я сказал все! Писать эту книгу об SQL Server 2000 было истинным удовольствием. Надеюсь, мне удалось привить и вам интерес к этой замечательной системе. Поэтому мой вам совет: ныряйте в нее, не колеблясь! Но помните: нельзя экспериментировать с работающей системой!

ПРИЛОЖЕНИЕ А

Установка Web-сервера

Чтобы наконец-то закончить разработку приложения Spy Net, на компьютере следует установить какой-либо Web-сервер. Установка Web-сервера поможет привести в действие интерфейс, разработанный в главе 12, "Разработка интерфейса пользователя базы данных SQLSpyNet". О том, как установить программу Personal Web Server (Личный Web-сервер) на компьютер, работающий под управлением операционной системы Windows 98, и пойдет речь в этом приложении.

Превратить компьютер в Web-сервер действительно не составляет никакого труда. Для этого необходимо всего лишь наличие программы Personal Web Server, которая имеется на инсталляционном компакт-диске Windows 98. При его отсутствии эту программу можно бесплатно загрузить с Web-узла компании Microsoft по адресу: `http://www.microsoft.com/msdownload/ntoptionpack/askwiz.asp`.

На заметку При установке программы *Personal Web Server* на компьютер, работающий под управлением операционной системы Windows NT, вам может пригодиться следующая информация.

- Руководство по установке дополнительных программ для Windows NT 4.0:

  `http://msdn.microsoft.com/library/periodic/period98/ewn98b1.htm`

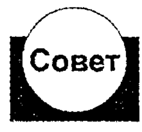
- Рекомендуемая информация от Microsoft относительно установки средства Internet Information Server 4.0 (для Windows NT):

  `http://support.microsoft.com/support/iis/install/install_iis4.asp?RLD=71`

Предположим, установочные файлы наконец-то найдены. Теперь осталось выполнить всего несколько простых шагов и... ликуйте: оказывается, все было так просто!

На заметку В этом приложении описывается стандартная установка Web-сервера. Другими словами, несмотря на то что при описании процесса установки SQL Server 2000 я затронул как тип установки *Typical* (Обычная), так и тип *Custom* (Выборочная), здесь я ограничусь только типом *Typical*. В конце концов, эта книга посвящена программе SQL Server 2000, а не разработке Web-страниц!

Когда для установки Web-сервера будет все готово, дважды щелкните на пиктограмме `Setup.exe`.



Если при помещении инсталляционного компакт-диска Windows 98 в накопитель CD-ROM программа установки не запускается автоматически, необходимый файл можно найти вручную. Программа установки для *Personal Web Server* находится в файле `x:\add-ons\pws\setup.exe`, где `x` — это накопитель CD-ROM на вашем компьютере.

После запуска программы установки на экране появится диалоговое окно, изображенное на рис. А.1.

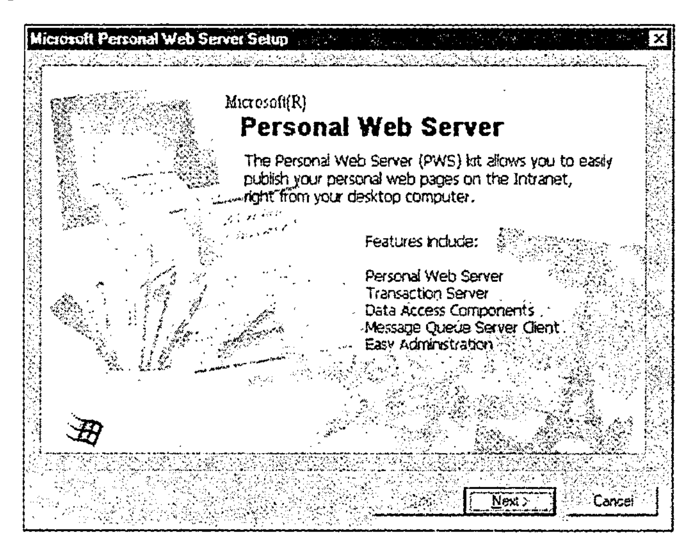


Рис. А.1. Начальное диалоговое окно программы установки Personal Web Server

Как и для большинства установочных программ, это диалоговое окно предлагает не слишком-то много возможностей. Все, что можно сделать на данном этапе, — это продолжить или же прекратить установку. Давайте все-таки продолжим. Щелкните на кнопке Next (Далее). На экране появится диалоговое окно с лицензионным соглашением, показанное на рис. А.2.

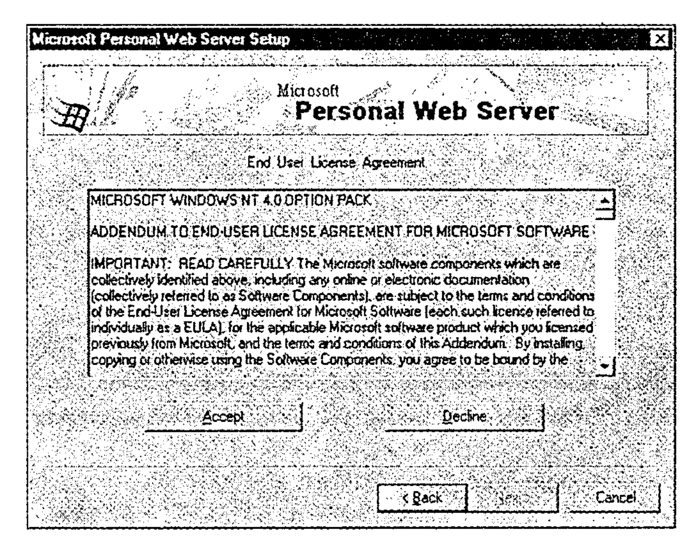


Рис. А.2. Диалоговое окно лицензионного соглашения для программы Personal Web Server

После принятия лицензионного соглашения (кнопка Accept (Принять)) на экране появится диалоговое окно, аналогичное показанному на рис. А.3.

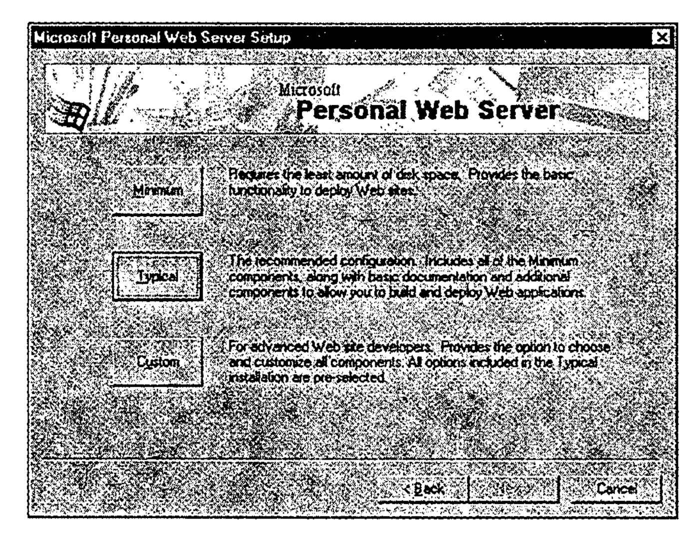


Рис. А.3. Возможные типы установки Personal Web Server

В этом окне вам предлагается выбрать желаемый тип установки: Minimum (Минимальная), Typical (Обычная) или Custom (Выборочная). Как уже упоминалось, рассмотрим только обычную установку. Щелкните на кнопке Typical, и на экране появится следующее диалоговое окно, позволяющее задать папки, в которых по умолчанию будут размещаться страницы Web-сервера (рис. А.4).

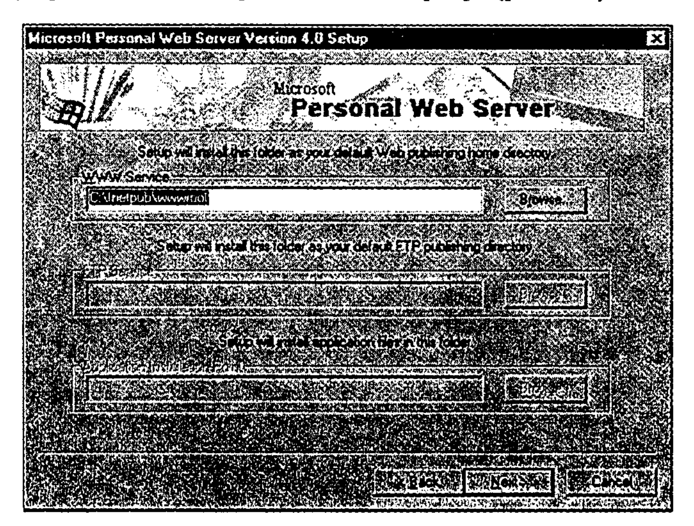


Рис. А.4. Выбор расположения Web-страниц будущего сервера



Как только Web-сервер заработает, все его страницы рекомендуется переместить в какое-нибудь другое место файловой системы. Это продиктовано в основном соображениями безопасности, поскольку каждый уважающий себя хакер первым делом не преминет звглянуть именно в папку `x:\Inetpub\wwwroot`.

Хотя для реальных систем расположение Web-страниц, выбранное по умолчанию, крайне не рекомендуется, в учебном примере это не так уж и важно, по-

этому на сей раз оставим все как есть. Щелкните на кнопке Next, и... вот, собственно говоря, и все! Программа установки начнет свою работу, а на экране появится окно, изображенное на рис. А.5.

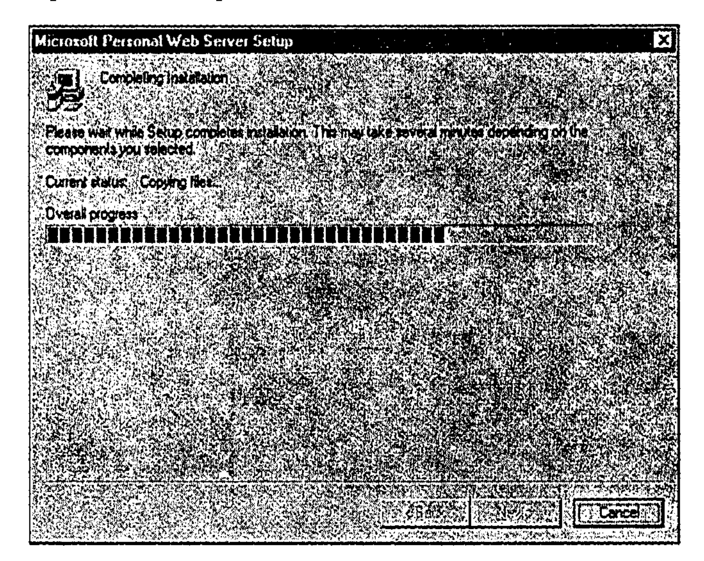


Рис. А.5. Процесс установки программы Personal Web Server

После окончания установки Web-сервера вам, скорее всего, придется перезагрузить компьютер. Ответив утвердительно на запрос установочной программы о необходимости перезагрузки компьютера, посмотрим, какие же программы устанавливаются вместе с Personal Web Server.

- *PWS Manager* (Диспетчер личного Web-сервера). Графический интерфейс пользователя, позволяющий настраивать службу Web.
- *Серверные расширения FrontPage* (FrontPage server extensions). Специальные файловые расширения, которые позволяют подключаться к Web-серверу и редактировать находящиеся на нем Web-страницы.
- *Компоненты доступа к данным Microsoft* (Microsoft Data Access Components — MDAC). Ключевые компоненты доступа к данным для SQL Server 2000, предназначенные для обмена информацией между не совместимыми друг с другом источниками данных.
- *Документация.* Файлы справки и другие документы, призванные облегчить знакомство с работой программы Personal Web Server.

Экскурс

Серверные расширения FrontPage

Обратили внимание на название *FrontPage*? Это не тот FrontPage, который мы привыкли использовать для создания Web-страниц. Серверные расширения FrontPage — это специальные расширения Web-сервера, которые позволяют подключаться к нему и редактировать находящиеся на сервере Web-страницы.

Вообще-то функции серверных расширений FrontPage этим не ограничиваются, однако некоторые сведения об их работе вы все же получили.

Между тем компьютер, должно быть, уже перезагрузился, поэтому попробуем убедиться в том, что Web-сервер действительно работает и, как говорится, "готов к употреблению".

Программу для настройки личного Web-сервера — Personal Web Manager — можно запустить из меню Start (Пуск), подменю Microsoft Personal Web Server (рис. A.6).

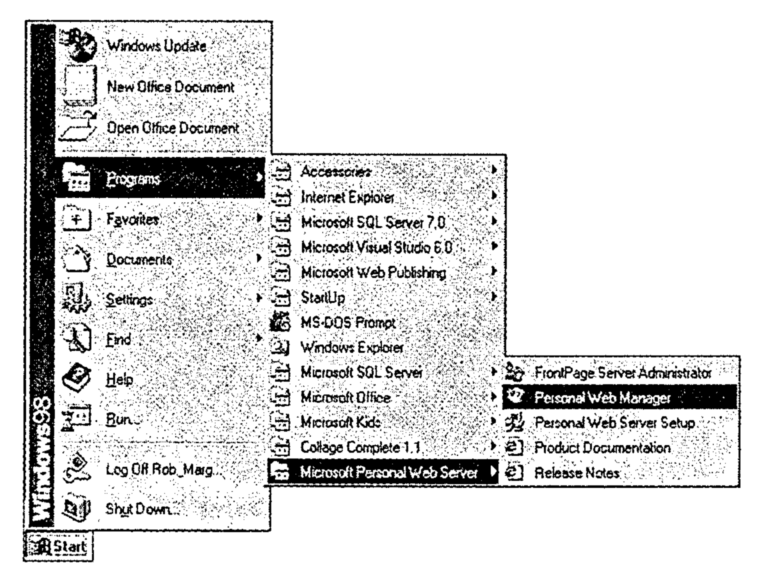


Рис. A.6. Один из способов запуска программы Personal Web Manager заключается в использовании главного меню

После запуска программы Personal Web Manager на экране появится диалоговое окно, изображенное на рис. A.7.

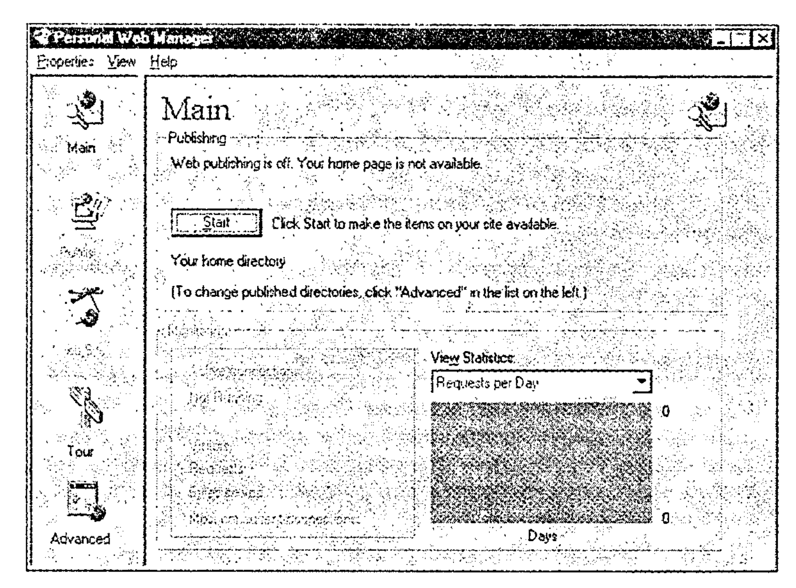


Рис. A.7. Программа PWS Manager и компоненты настройки Web-сервера

На заметку Как правило, первым диалоговым окном, открывающимся при запуске программы *Personal Web Manager*, является окно подсказки ("совета дня"). Совет дня обычно содержит весьма ценную информацию, призванную облегчить знакомство с *PWS Manager*.

Главное окно настройки содержит кнопку-переключатель Start/Stop (Запустить/Остановить). Если Web-служба работает, кнопка будет находиться в состоянии Stop, и наоборот. Поскольку в данном случае кнопка находится в состоянии Start, щелкните на ней — и вот вам готовый Web-сервер (рис. A.8)!

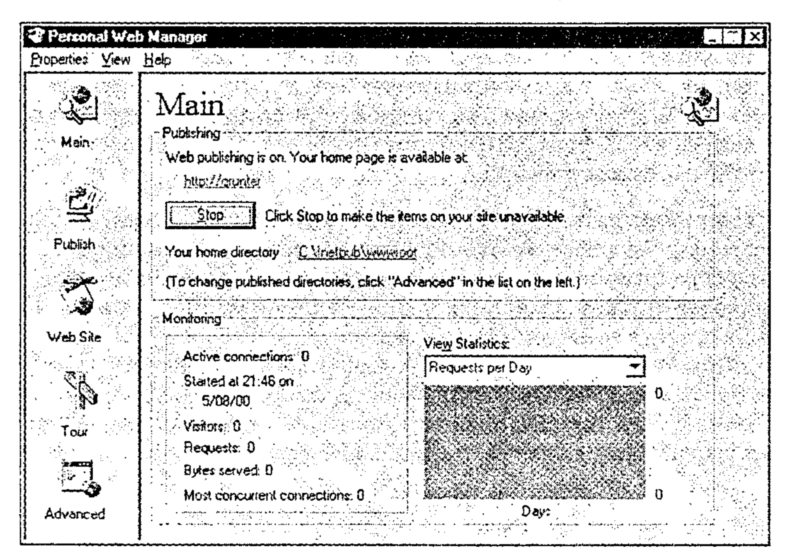


Рис. A.8. Назначенная по умолчанию домашняя страница вашего Web-сервера

Чтобы проверить настройку домашней страницы, запустите обозреватель Internet и введите имя вашего Web-сервера, например `http://grunter` или `http://127.0.0.1`. Отображаемая домашняя страница относится непосредственно к самому Web-серверу, а не к Web-интерфейсу приложения SQLSpyNet. Чтобы узнать, как превратить приложение SQLSpyNet в Web-узел (под названием Spy Net), обратитесь к главе 12, "Разработка интерфейса пользователя базы данных SQLSpyNet".

Итак, свершилось! Теперь ваш компьютер не просто какой-то там SQL-сервер, он стал еще и Web-сервером!

Кстати, если это вас интересует, чтобы настроить компьютер как Web-сервер, вовсе не обязательно подключаться к Internet. С такой конфигурацией, как у нас, мы получаем скорее узел сети intranet, нежели Internet.

Термин *Узел сети Internet* (*Internet site*) — это узел, открытый для публичного доступа с любого компьютера, имеющего доступ к Internet, как, например, `http://www.williamsblishing.com` или `http://www.quepublishing.com`.

Термин В отличие от предыдущего, *узел сети intranet* (*intranet site*) — это защищенный узел с ограниченным доступом. К примеру, большинство компаний используют intranet для того, чтобы держать своих сотрудников в курсе последних событий. Создаваемый узел Spy Net также относится

к сети intranet. Основной характеристикой intranet является принадлежность всех ее терминалов к локальной сети. Другими словами, это означает, что все компьютеры такой сети физически расположены в непосредственной близости друг от друга.

Термин Сети *extranet* подобны intranet в том смысле, что доступ к ним ограничен, однако, в отличие от intranet, они размещаются не в локальной сети, а в глобальной. Это значит, что компания, например такая, как Spy Net, может работать с общим узлом, даже если один из ее офисов находится в Лондоне, а другой, скажем, в Бостоне. Кроме сотрудников компании, к extranet в принципе могут получать доступ и другие люди, скажем, поставщики, потребители или привилегированные клиенты. В этом случае для входа в сеть компании используется узел Internet, для которого предварительно устанавливаются крайне ограниченные права доступа.

Поскольку наш компьютер не обязательно может быть подключен к Internet, для работы с узлом Spy Net вовсе не нужно устанавливать соединение с каким-либо из поставщиков услуг Internet. Только подумайте, теперь у вас есть собственные Web-сервер, SQL-сервер и даже локальная сеть — прямо дух захватывает! Может быть, вскоре вы даже станете сетевым инженером, а?

Ну вот, собственно говоря, и все; теперь самое время вернуться к главе 12, "Разработка интерфейса пользователя базы данных SQLSpyNet", и наконец-то закончить установку узла Spy Net. А потом поскорее займитесь своей карьерой Web-разработчика (не забывая, конечно, и об администрировании баз данных).

Приложение Б

Установка и настройка SQL Server 2000

В этом приложении...

Данное руководство по установке основано на информации с инсталляционного компакт-диска SQL Server 2000. В этом приложении описывается самый надежный способ установки и настройки SQL Server 2000. Запомните, это вовсе не единственный возможный способ, просто он наиболее оптимально удовлетворяет всем требованиям.

Данное приложение содержит много рисунков и пошаговых инструкций, которые наглядно описывают практически все, что нужно знать для успешного проведения установки.

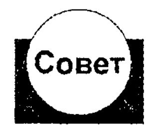
На заметку Обратите внимание на то, что описанная здесь установка SQL Server 2000 проводилась на отдельно взятом компьютере, т.е. на компьютере, не подключенном к сети и не настроенном на работу с сетью. Поэтому, если ваша система подключена к сети или же на ней установлены сетевые драйверы, диалоговые окна, описанные в этом приложении, могут отличаться от тех, что вы увидите на своем экране.

Предполагается, что перед началом установки у вас уже есть следующее.

- Необходимый минимум программного обеспечения. Более подробно эти требования описаны во введении.
- SQL Server 2000 Personal Edition. Именно эта версия SQL Server 2000 используется для разработки приложения SQLSpyNet. Тем не менее, если вы имеете SQL Server 2000 Standard Edition или же работаете с сетью, оставьте все как есть и не удивляйтесь, если в процессе установки обнаружатся некоторые отличия в диалоговых окнах.
- Инсталляционный компакт-диск операционной системы Windows 98 или другой версии Windows (на всякий случай).

На заметку Большая часть процесса установки SQL Server 2000 Personal Edition происходит абсолютно одинаково для Windows 98, Windows 2000 и Windows NT. Здорово, правда? Ну а в случае каких-либо различий я обязательно о них расскажу.

Кроме того, надеюсь, вы уже создали резервные копии всех важных файлов, хранящихся на компьютере, чтобы все можно было поправить, если установка вдруг пойдет "наперекосяк". Следует отметить, что установки подобных программ, как правило, не требуют значительного вмешательства со стороны пользователя, однако осторожность все же не помешает.



Перед началом описанного здесь процесса установки мой компьютер уже был оснащен одним *экземпляром* SQL Server 2000 (я еще не раз скажу об этом). Тем не менее вы можете использовать инструкции, приведенные здесь, вне зависимости от того, установлены ли уже экземпляры SQL Server 2000 (либо предыдущих версий) или еще нет. По ходу установки я буду сообщать о различиях, когда таковые будут встречаться, однако принцип установки во всех случаях один и тот же.

Термин

Экземпляром (*instance*) SQL-сервера называется копия приложения SQL Server, установленная на компьютере. В принципе это может быть любая версия SQL-сервера, однако если она более старая, чем SQL Server 2000, то новому экземпляру будет присвоено имя `Default`. Именованные экземпляры позволяют устанавливать на одном компьютере несколько копий SQL-сервера (новая возможность), каждая из которых работает, как отдельный компьютер.

Итак, кажется, все готово... Ну что ж — поехали!

Выбор типа установки

Поместите инсталляционный компакт-диск SQL Server 2000 в накопитель CD-ROM. Программа установки должна запуститься автоматически, если же этого не произойдет, найдите в корневом каталоге компакт-диска файл `Autorun.exe` и дважды щелкните на нем, чтобы начать процесс установки.

В результате запуска программы установки на экране появится диалоговое окно, подобное показанному на рис. Б.1.

На этом этапе вам предлагается выбрать одну из пяти опций. Рассмотрим наиболее интересные из них.

Опция SQL Server Prerequisites (Компоненты, необходимые для работы с SQL Server 2000) позволяет установить обновления библиотеки стандартных элементов управления (Common Controls Library) для Windows 95 (в случае, если был выбран тип установки *Connectivity only* (Только средства связи)).

На заметку

Тип установки *Connectivity only* позволяет клиенту, работающему под управлением Windows 95, "общаться" с SQL Server 2000. Данный тип установки *не включает* в себя никаких инструментов управления, которые используются для настройки SQL Server 2000. Такая установка применяется в основном для того, чтобы позволить прикладным программам пользователей, написанным для операционной системы Windows 95, устанавливать соединение с SQL Server 2000.

Еще одна опция, а именно Browse Setup/Upgrade Help (Обзор справки по установке/обновлению), позволяет получить справку по проведению установки для конкретной операционной системы, а также по возможностям установки, доступным для определенных платформ.

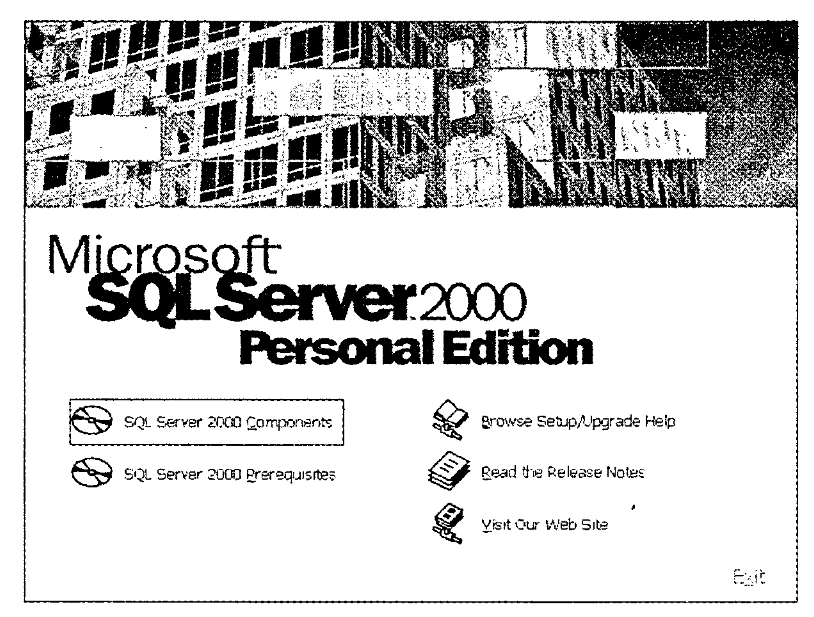


Рис. Б.1. Начальное диалоговое окно программы установки для SQL Server 2000

Чтобы продолжить установку, выберите опцию SQL Server 2000 Components (Компоненты SQL Server 2000). На экране появится новое диалоговое окно, чрезвычайно похожее на предыдущее, однако с меньшим количеством опций. Это окно позволяет установить ядро базы данных и сопутствующие службы для SQL Server 2000 (рис. Б.2).

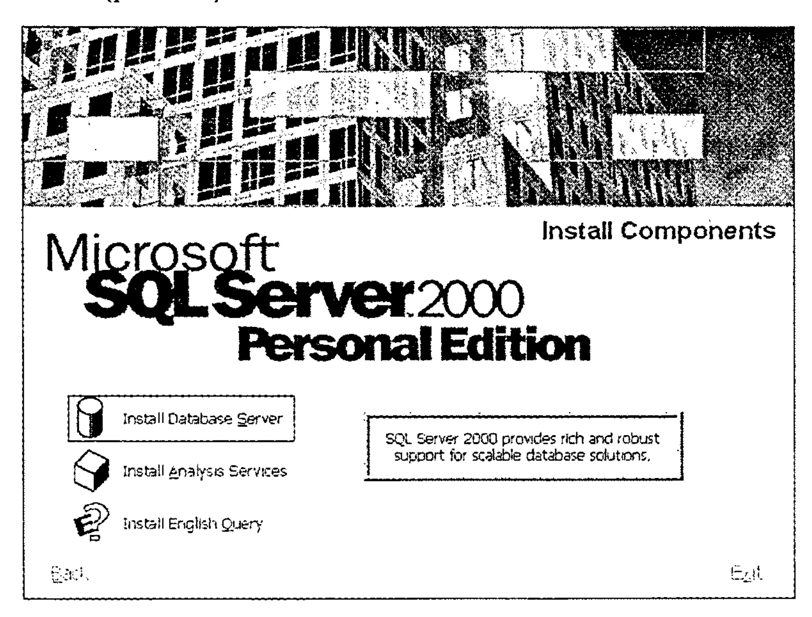


Рис. Б.2. Установка ядра базы данных для SQL Server 2000 Personal Edition

- Служба Analysis Services (Служба анализа) заменила собой компоненты аналитической обработки данных (Online Analytical Processing — OLAP), которые присутствовали в версии SQL Server 7.0. Служба анализа позволяет заниматься разработкой различных сервисов OLAP и компонентов для извлечения данных.
- Средство English Query (Запросы на естественном языке) предназначено для дальнейшей работы со службами SQL Server 2000. Это средство позволяет пользователям формулировать запросы к серверу баз данных не на языке SQL, а в виде предложений естественного (английского) языка.

Пока что не стоит уделять этим службам слишком много внимания, поскольку для разработки нашего приложения они не понадобятся. В случае необходимости о них можно прочитать в справке, которая поставляется вместе с мастером установки (см. рис. Б.1).

Чтобы наконец-то начать установку SQL Server 2000 Personal Edition, выберите опцию Install Database Server (Установка сервера баз данных). На экране должно появиться окно, изображенное на рис. Б.3.

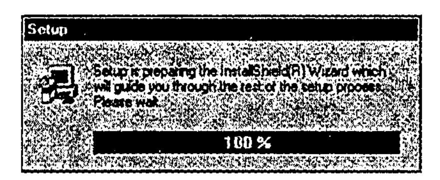


Рис. Б.3. Запуск мастера установки для приложения SQL Server 2000 Personal Edition

Выбор этой опции приведет к запуску мастера установки (загрузка которого может занять некоторое время). Пошаговое выполнение установки с помощью этого мастера рассматривается в следующих разделах.

Установка приложения с помощью пошагового мастера установки

Начальное диалоговое окно мастера установки под названием Welcome (Добро пожаловать) изображено на рис. Б.4. Как обычно, возможностей для выбора на этом этапе не так уж много. Если вы уверены в необходимости продолжения процесса установки, щелкните на кнопке Next (Далее) — вот пока и все.

Следующее диалоговое окно Computer Name (Имя компьютера) показано на рис. Б.5.

На заметку Если установка проводится на компьютере, настроенном на работу с сетью, в этом диалоговом окне вам будет позволено выбрать и другие опции (а не только *Local Computer* (Локальный компьютер), как показано на рис. Б.5). Если компьютер подключен к сети, не забудьте еще раз убедиться в том, что вы выбрали опцию *Local Computer*. Чтобы продолжить установку, щелкните на кнопке *Next*.

Третье диалоговое окно Installation Selection (Выбор установки) показано на рис. Б.6. Если в будущем, уже после установки экземпляра SQL Server 2000, вам понадобится модифицировать или удалить какие-либо его компоненты, можно воспользоваться опцией Upgrade, remove or add components to an existing instance of SQL Server 2000 (Обновить, добавить или удалить компоненты существующего экземпляра SQL Server 2000).

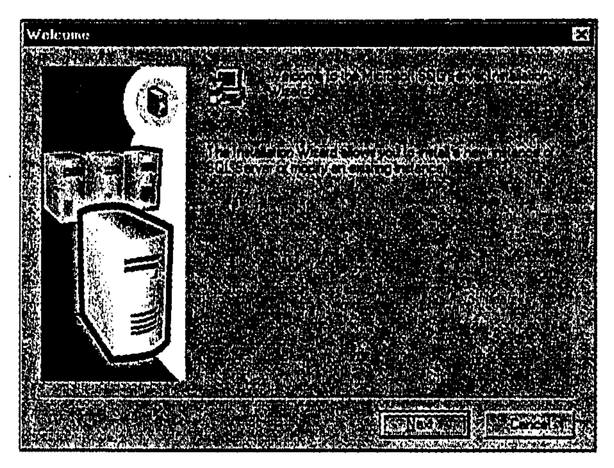


Рис. Б.4. Диалоговое окно Welcome мастера установки SQL Server 2000

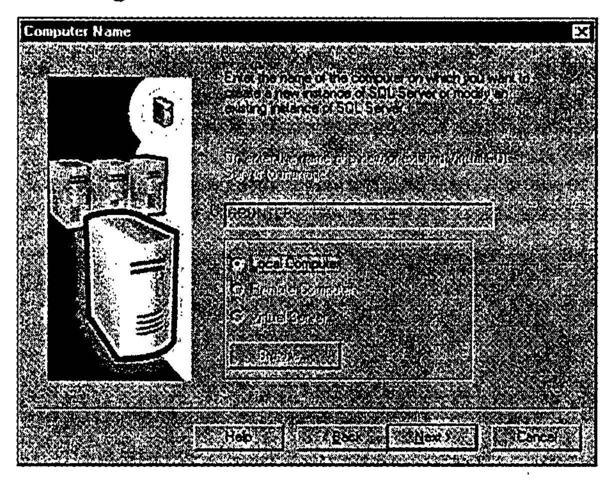


Рис. Б.5. Выберите компьютер, на котором необходимо установить экземпляр SQL Server 2000

Выбор типа установки Advanced Options (Дополнительные возможности) заключает в себе следующие возможности.

- Record an Unattended .ISS file (Создать ISS-файл для автоматической установки). Позволяет создать установочный файл для автоматической установки, т.е. не требующей вмешательства пользователя.
- Registry Rebuild (Восстановление системного реестра). Предназначена для обнаружения и исправления поврежденной информации системного реестра о данном экземпляре SQL Server 2000.
- Maintain a Virtual Server for Fail-over Clustering (if your previous installation supports it) (Установка виртуального сервера для управления отказоустойчивой кластеризацией (если последняя поддерживалась в предыдущей установке)). Позволяет вносить изменения в существующие кластерные настройки, например изменение имени, добавление или удаление узлов. Если предыдущая установка SQL Server на вашем компьютере не поддерживает таких возможностей, данная опция будет недоступна.

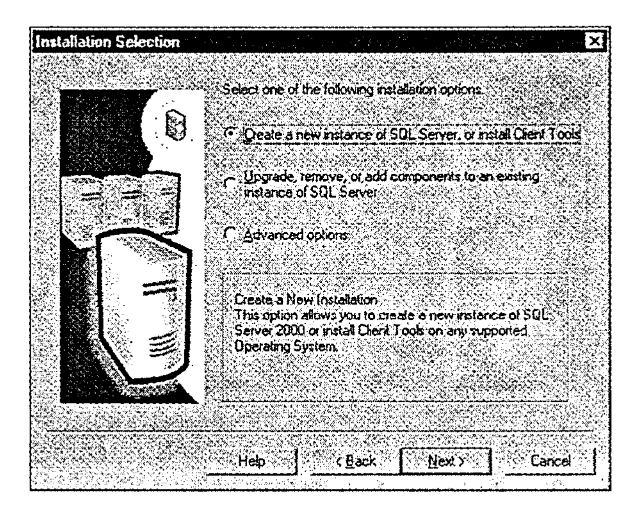


Рис. Б.6. Диалоговое окно Installation Selection мастера установки SQL Server 2000

Однако, если вы еще не забыли, мы собираемся устанавливать новый экземпляр SQL Server 2000 Personal Edition. Поэтому выберите опцию Create a new instance of SQL Server, or install Client Tools (Создать новый экземпляр SQL Server или установить компоненты клиента) и щелкните на кнопке Next.

Следующее диалоговое окно мастера установки под названием User Information (Информация о пользователе) представлено на рис. Б.7.

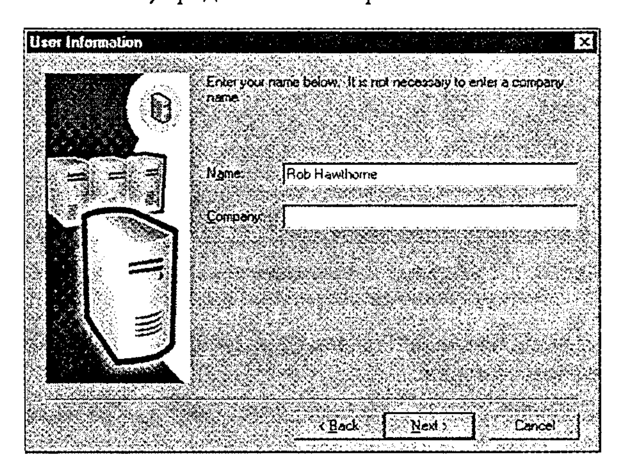


Рис. Б.7. Диалоговое окно User Information мастера установки SQL Server 2000

Здесь следует указать информацию, которая будет использована при регистрации установки. Введите в соответствующие поля диалогового окна свое имя и, если это необходимо, сведения о компании, а затем щелкните на кнопке Next.

На экране появится диалоговое окно с лицензионным соглашением о программном обеспечении (Software License Agreement) и поправками к лицензионному соглашению конечного пользователя (End User License Agreement — EULA). Чтобы продолжить процесс установки, необходимо щелкнуть на кнопке Yes (Да), как показано на рис. Б.8.

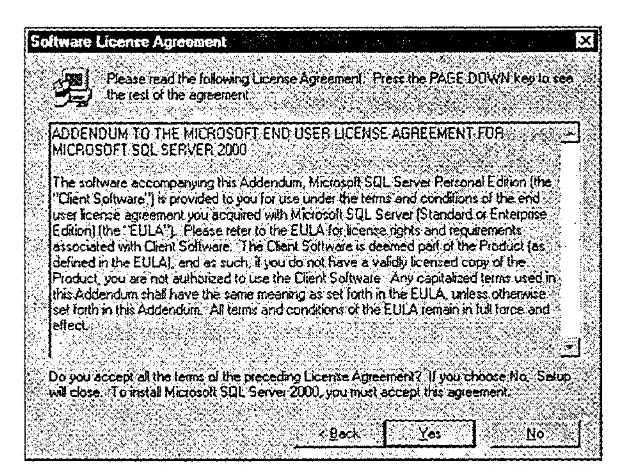


Рис. Б.8. Диалоговое окно Software License Agreement мастера установки SQL Server 2000

Следующее окно мастера установки под названием Installation Definition (Выбор компонентов установки) представлено на рис. Б.9.

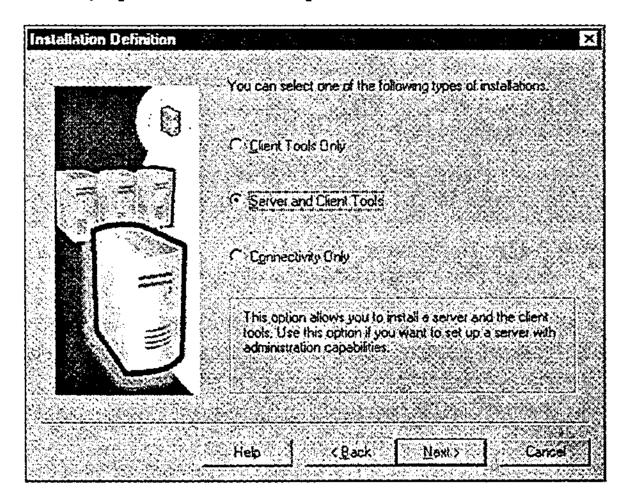


Рис. Б.9. Диалоговое окно Installation Definition мастера установки SQL Server 2000

В этом окне следует выбрать одну из предлагаемых опций.

- Client Tools Only (Только компоненты клиента). Используйте эту опцию для того, чтобы установить только компоненты, необходимые для подключения к существующему серверу, а также если хотите получить возможность выбора отдельных компонентов.
- Server and Client Tools (Компоненты сервера и клиента). Как раз то, что нам нужно. Эта опция позволяет установить сервер с административными ресурсами и дает доступ ко всему набору возможностей установки.
- Connectivity Only (Только средства связи). Используйте эту опцию в том случае, если вам необходимо всего лишь подключаться к существующему серверу, но не требуется возможность выбора отдельных компонентов. Опция Connectivity Only подра-

зумевает только установку файлов, необходимых для подключения клиентских приложений к серверу. В ходе этой установки компоненты доступа к данным Microsoft (Microsoft Data Access Components — MDAC), например, обновляются таким образом, что клиент получает возможность подключаться к SQL Server с помощью ODBC. Не волнуйтесь, если вы чего-то не поняли; мы еще вернемся к этой теме несколько позднее, когда будем обсуждать интерфейсные прикладные программы.

Чтобы продолжить установку, выберите опцию Server and Client Tools и щелкните на кнопке Next.

В следующем диалоговом окне Instance Name (Имя экземпляра) можно присвоить имя устанавливаемому экземпляру SQL Server 2000 (рис. Б.10).

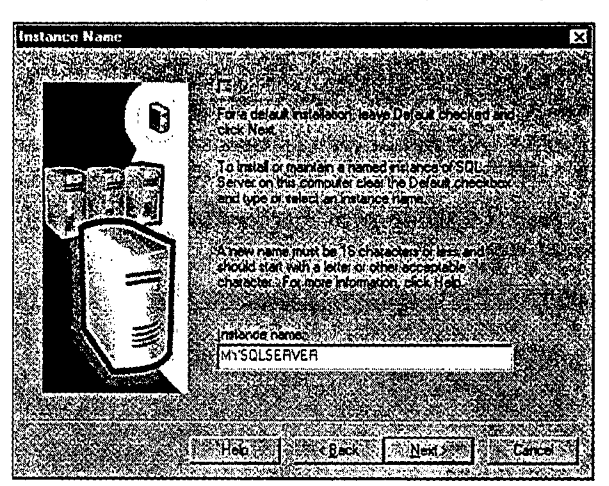
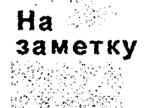


Рис. Б.10. В данном случае при установке SQL Server 2000 опция Default недоступна

При выполнении установки, описанной в этой главе, я назвал свой новый экземпляр SQL Server 2000 `MYSQLSERVER`. Естественно, вы вовсе не обязаны использовать именно это имя; называйте свой экземпляр как будет душе угодно. Все, что требуется от имени экземпляра, — это соответствие нескольким правилам (они будут рассмотрены немного позднее).

Опция Default (По умолчанию), расположенная в верхней части диалогового окна, позволяет присвоить данному экземпляру имя, принятое по умолчанию для службы, под управлением которой работает SQL Server 2000.



На заметку Если опция *Default* недоступна (см. рис. Б.10), это значит, что на вашем компьютере уже существует минимум один экземпляр SQL Server. Это может быть экземпляр SQL Server 6.5, 7.0 или 2000. Как видите, я также не могу установить флажок *Default*, потому что на моем компьютере уже имеется экземпляр SQL Server.

Как уже отмечалось, имя экземпляра должно удовлетворять нескольким правилам, перечисленным ниже. Эти правила взяты из справки по установке SQL Server, которая содержится на инсталляционном компакт-диске.

- В имени экземпляра регистр букв не учитывается.
- В качестве имен нельзя использовать зарезервированные слова `Default` и `MSSQLServer`.

- Количество букв в имени не должно превышать 16.
- Первым символом имени может быть буква, знак &, символы подчеркивания (_) или номера (#). Допустимые буквы определяются согласно стандарту Unicode 2.0 и включают латинские буквы A-Z и a-z, а также буквы национальных языков.
- Последующие символы имени могут быть такими:
  - Буквы, определенные стандартом Unicode 2.0;
  - Десятичные цифры латинского и других национальных алфавитов;
  - Символы $, # или _;
  - Не допускается использование в имени экземпляра пробелов, специальных символов, а также обратной косой черты (/), запятой (,), двоеточия (:) или знака @.



В именах экземпляров SQL Server 2000 могут использоваться только те символы, которые поддерживаются текущей кодовой страницей Microsoft Windows. Если в имени будет использован символ Unicode, который не поддерживается текущей кодовой страницей, появится сообщение об ошибке.

Даже если опция Default доступна, советую не устанавливать этот флажок. Лучше снимите его и введите свое имя экземпляра (желательно то же, что использовал я). Почему? Дело в том, что одно из наиболее существенных нововведений SQL Server 2000 — это возможность устанавливать на одном компьютере несколько экземпляров SQL Server 2000 одновременно. Ну а кроме того, это имя отлично нам подходит!



Если вы все-таки решите установить экземпляр по умолчанию, диалоговые окна, приведенные в этом приложении, возможно, будут несколько отличаться от тех, которые вы увидите на своем экране.

Экземпляр, экземпляр... Полцарства за экземпляр!

Экскурс

В предыдущих версиях SQL Server на одном компьютере разрешалось иметь только один его экземпляр (который теперь называется `Default`). Это означало, что, если для нужд разработки или тестирования вам был необходим еще один экземпляр SQL Server, для его установки приходилось привлекать еще один компьютер. Но теперь этому пришел конец!

С появлением SQL Server 2000 на одном компьютере позволено иметь несколько экземпляров системы управления реляционными базами данных в сочетании с экземплярами предыдущих версий (как, например, на моем компьютере). Все экземпляры представляют собой отдельные программы, поэтому каждый из них владеет собственным "маленьким кусочком" компьютера, включая память и пространство на жестком диске.

Между тем, как это часто случается, слишком много хорошего тоже не бывает. Специалисты Microsoft рекомендуют не устанавливать на одном компьютере больше 16 экземпляров SQL Server 2000 одновременно (это то количество, которое они протестировали, включая и `Default`); более того, предлагают по возможности ограничиться только одним. Но как все-таки греет душу мысль, что мы можем получить больше, если только захотим!

На заметку Если вы не устанавливаете экземпляр SQL Server 2000 `Default` сейчас, это можно будет сделать и потом, в любое время.

После того как вы ввели имя нового экземпляра, щелкните на кнопке Next. Если вы попытаетесь щелкнуть на этой кнопке, не указав предварительно имя экземпляра и не установив флажок Default, на экране появится сообщение об ошибке. В нем будет сказано о необходимости обязательного определения имени экземпляра.

Следующее диалоговое окно мастера установки Setup Type (Тип установки) показано на рис. Б.11.

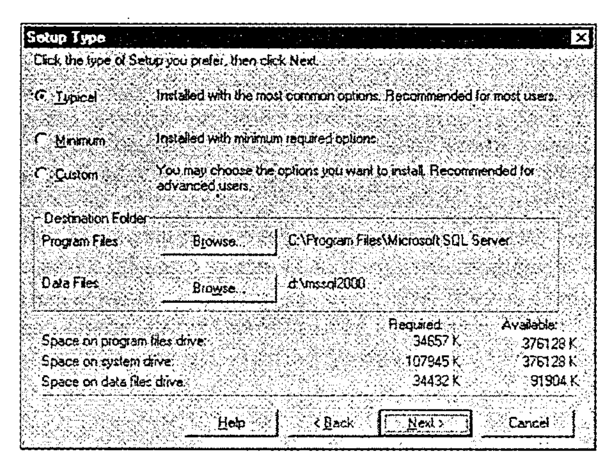


Рис. Б.11. Диалоговое окно Setup Type мастера установки SQL Server 2000

В этом окне можно выбрать один из трех типов установки.

- Typical (Обычная). Рекомендуется для большинства пользователей. При этом на компьютер устанавливаются все средства и дополнения SQL Server 2000, кроме компонента Code Samples (Примеры кода). Из средств разработки Development Tools устанавливается только средство отладки. Для выполнения упражнений, описанных в этой книге, такой тип установки будет в самый раз.
- Minimum (Минимальная). Устанавливаются только ключевые компоненты, необходимые для работы SQL Server 2000. Такая установка особенно эффективна, если на жестком диске не хватает места.
- Custom (Выборочная). Позволяет самостоятельно выбрать компоненты и дополнения, которые должны быть установлены на компьютер. Кроме того, эта опция позволяет изменить параметры сортировки Windows и SQL Server 2000. Параметры сортировки включают набор используемых символов национального алфавита и порядок их сортировки. Используйте эту опцию в том случае, если вы абсолютно точно представляете себе, какая конфигурация вам нужна.

Кнопки Browse (Обзор) этого диалогового окна позволяют указать папки, в которых SQL Server 2000 будет хранить данные и программные файлы, что может пригодиться в случае, если у вас есть проблемы с наличием свободного места на определенных дисках. Если же с этим все в порядке, можете оставить настройки, выбранные по умолчанию.

На заметку

Для нашей установки следует выбрать опцию *Typical*. Однако, если по каким-либо причинам вам действительно необходимо изменить параметры сортировки или выбрать устанавливаемые дополнения, можете воспользоваться и опцией *Custom*. Не волнуйтесь, это вовсе не так страшно, как кажется; более подробно выборочная установка будет рассмотрена далее в приложении.

После того как вы выберете опцию Typical и щелкнете на кнопке Next, на экране появится диалоговое окно, показанное на рис. Б.12.

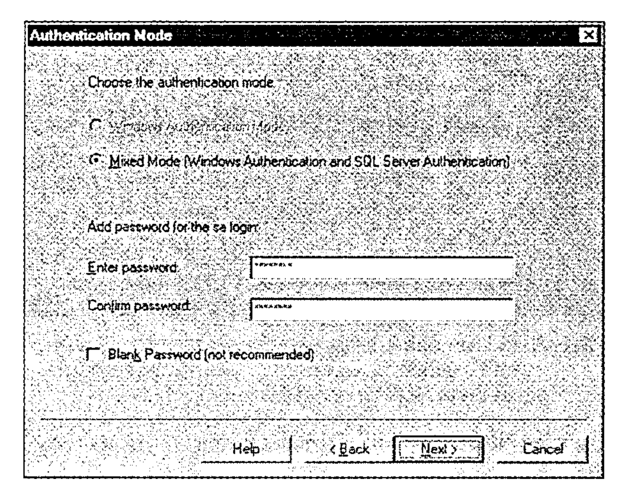


Рис. Б.12. Диалоговое окно Authentication Mode мастера установки SQL Server 2000

Диалоговое окно Authentication Mode (Режим аутентификации) позволяет задать пароль учетной записи пользователя для системного администратора (System Administrator — sa) SQL Server 2000. Убедитесь в том, что заданный пароль не так легко забыть (разумеется, вам) и не так легко отгадать (разумеется, другим).



Хорошо запомните (а еще лучше запишите и спрячьте в надежном месте) пароль учетной записи системного администратора. Если вы забудете этот пароль, вам, к сожалению, ничего другого не останется, как удалить и снова переустановить SQL Server 2000.

На заметку

Если вы подключены к сети либо работаете под управлением операционной системы Windows 2000 или Windows NT, то в диалоговом окне *Authentication Mode*, возможно, будет доступна и опция *Windows Authentication Mode* (Режим аутентификации Windows). Этот режим означает, что в качестве учетной записи, которая будет использоваться службой SQL для предоставления доступа к экземпляру SQL Server 2000, выступит учетная запись текущего пользователя, зарегистрировавшегося в системе. Если эта опция будет доступной, рекомендую выбрать не ее, а опцию *Mixed Mode* (Смешанный режим). Это позволит вам и подключаться к SQL Server как системному администратору, и устанавливать соединение через вашу сеть на базе Windows NT/2000 (если таковая имеется). Более подробно о безопасности речь идет в главе 9, "Обеспечение безопасности базы данных Spy Net".

После того как вы в очередной раз щелкнете на кнопке Next, на экране появится диалоговое окно Start Copying Files (Начать копирование файлов), показанное на рис. Б.13. Оно предоставляет вам последнюю возможность до начала установки вернуться назад и изменить настройки или же отменить их совсем.

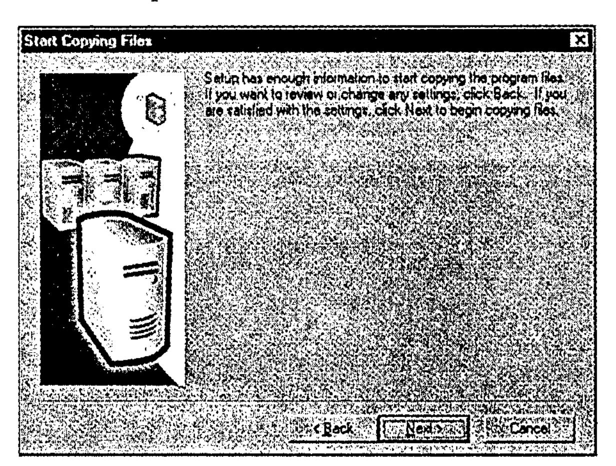


Рис. Б.13. Начните копирование файлов или же вернитесь назад и измените настройки процесса установки SQL Server 2000

Щелкните на кнопке Next для того, чтобы начать установку приложения с выбранными настройками.

Теперь можете немного расслабиться и пойти выпить чашечку кофе, пока процесс установки скопирует несколько (хотя, по-моему, их там сотни!) файлов с инсталляционного компакт-диска на жесткий диск вашего компьютера. Все это время на экране, скорее всего, будет "висеть" окно, изображенное на рис. Б.14.

Когда все это наконец-то закончится (приблизительно 15 минут на моем Pentium 75), позвольте программе установки перезагрузить компьютер (если она этого требует) и для полного счастья щелкните на кнопке Finish (Закончить) в диалоговом окне Setup Complete (Установка завершена), как показано на рис. Б.15. Не правда ли, это было совсем не больно?

Что случится, если выбрать тип установки *Custom*

Если в качестве типа установки вы все-таки выбрали Custom (Выборочная), смею предположить, что либо вам хочется получить от SQL Server 2000 все возможное, либо у вас действительно есть необходимость поменять параметры сортировки. Тип установки Custom подразумевает выполнение большего числа шагов, чем при только что рассмотренном типе установки Typical.

Первое отличие — это появление на экране диалогового окна Select Components (Выбор компонентов), показанного на рис. Б.16.

Данное окно позволяет самостоятельно выбрать компоненты, которые не были включены в тип установки Typical (такие, как примеры кода и некоторые дополнения), а также удалить компоненты, которые вам не нужны.

Рис. Б.14. Копирование файлов на жесткий диск в процессе установки SQL Server 2000

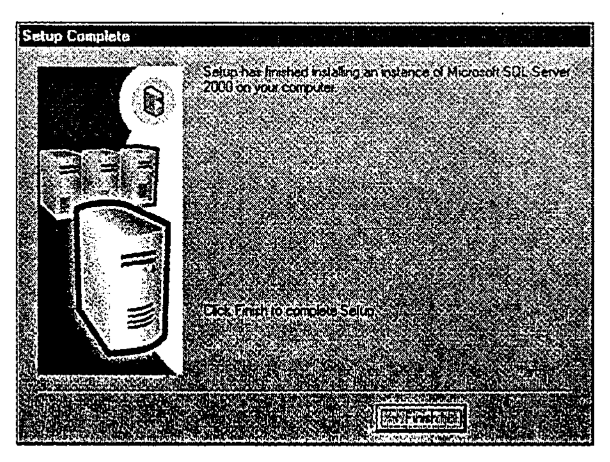


Рис. Б.15. Последнее диалоговое окно мастера установки SQL Server 2000



Будьте предельно внимательны при удалении компонентов. Если вы случайно удалите компонент, необходимый для работы SQL Server 2000, вся установка пойдет насмарку и вам придется начинать все сначала.

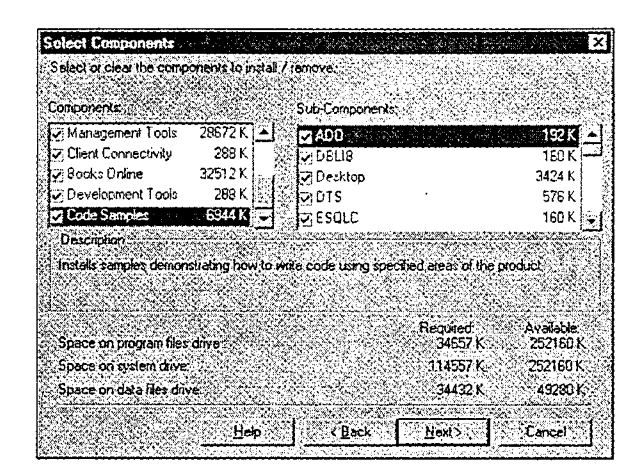


Рис. Б.16. Выберите компоненты SQL Server 2000, которые следует установить

Как уже отмечалось, с помощью этого диалогового окна вы можете добавить к установке образцы программных кодов, которые поставляются вместе с SQL Server 2000 и могут пригодиться в процессе разработки клиентских приложений для работы с базой данных. Эта опция почему-то очень хитро запрятана в самом низу списка компонентов, который находится в левой части окна Select Components. Чтобы выбрать эту опцию, установите соответствующий ей флажок (см. рис. Б.16).

Когда наконец-то вы останетесь довольны выбранными компонентами, щелкните на кнопке Next, чтобы продолжить установку.

На экране появится диалоговое окно Authentication Mode (рис. Б.17).

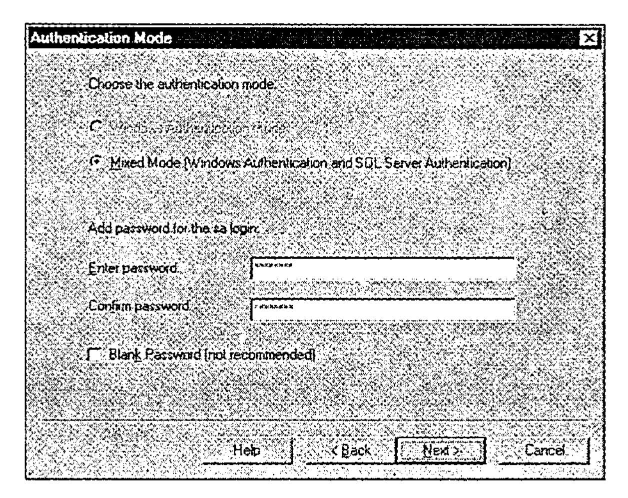


Рис. Б.17. Диалоговое окно Authentication Mode мастера установки SQL Server 2000

Данное диалоговое окно позволяет задать пароль учетной записи для системного администратора SQL Server 2000. Задавая пароль, убедитесь, что вам его будет легко запомнить, а всем остальным — трудно отгадать.



Хорошо запомните (а еще лучше запишите и спрячьте в надежном месте) пароль учетной записи системного администратора. Если вы забудете этот пароль, вам, к сожалению, ничего другого не останется, как удалить и снова переустановить SQL Server 2000.

На заметку Если вы подсоединены к сети или же работаете под управлением операционной системы Windows 2000 или Windows NT, то в диалоговом окне *Authentication Mode* вам, возможно, будет доступна и опция *Windows Authentication Mode* (Режим аутентификации Windows). Этот режим означает, что в качестве учетной записи, которая будет использоваться службой SQL для предоставления доступа к экземпляру SQL Server 2000, выступит учетная запись текущего пользователя, вошедшего в систему. Если эта опция будет доступной, рекомендую выбрать не ее, а опцию *Mixed Mode* (Смешанный режим). Это позволит вам и подключаться к SQL Server как системному администратору, и устанавливать соединение через вашу сеть Windows NT/2000 (если таковая имеется). Более подробно о безопасности речь идет в главе 9, "Обеспечение безопасности базы данных Spy Net".

Следующее диалоговое окно под названием Collation Settings (Параметры сортировки) представлено на рис. Б.18.

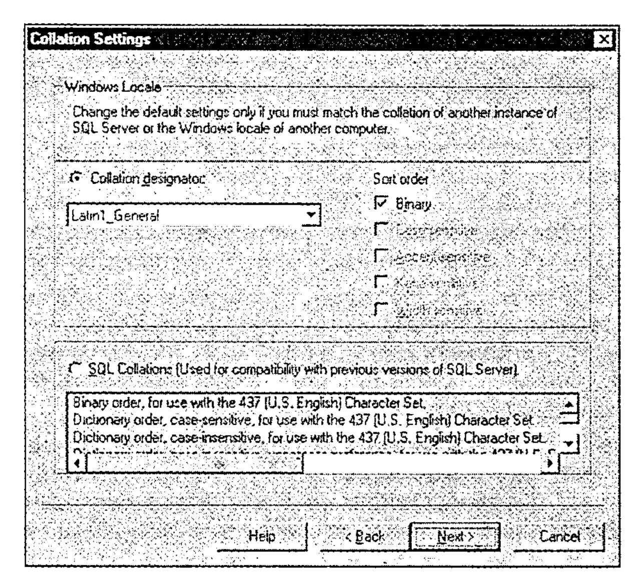


Рис. Б.18. Диалоговое окно с параметрами сортировки типа Custom приложения SQL Server 2000

Вот тут-то и начинается самое интересное. Для того чтобы "общаться" с другим экземпляром SQL Server, *кодировка* Windows или параметры сортировки которого отличаются от установленных на вашем компьютере, необходимо "подогнать" свои параметры под параметры этого экземпляра, который может быть установлен как на удаленном компьютере, так и на вашей машине. Для разработки приложения SQL Spy Net нам это, конечно же, не потребуется, но если вдруг нужда прихватит...

Термин *Кодировка* (*locale*) — это способ хранения символов в памяти компьютера и отображения их на экране. Кодировка компьютера выбирается во время установки операционной системы. В разных странах используются различные кодировки символов, позволяющие работать с буквами алфавита того или иного национального языка. Например, для работы с набором символов английского языка (США) используется кодировка *Latin1_General*.

На заметку Если вы подключаетесь к компьютеру, который использует кодировку, отличную от вашей, то при передаче данных от одного компьютера к другому компьютер-получатель должен преобразовывать полученные данные в свою кодировку. Это может привести к тому, что передача данных будет осуществляться очень медленно, особенно если речь идет о больших объемах информации. Если же компьютер-отправитель использует символы, которые не входят в набор символов кодировки компьютера-получателя (расширенный набор символов), то данные будут попросту утеряны.

После выбора кодировки следует убедиться в правильной настройке параметра Sort Order (Порядок сортировки). Описанные ниже опции определяют различные правила и комбинации правил сортировки.

- Binary (Бинарный). Наиболее быстрый порядок сортировки, при котором всегда учитывается регистр. Это значит, что строчные символы при сортировке всегда будут предшествовать прописным. Так, например, строчная "м" всегда будет стоять перед прописной "М". Если выбрана опция Binary, опции Accent sensitive (Учитывать ударение) и Case sensitive (Учитывать регистр) становятся недоступными.
- Case sensitive (Учитывать регистр). При выборе этой опции SQL Server 2000 различает буквы верхнего и нижнего регистра. Как и в предыдущем случае, при сортировке строчные символы всегда будут предшествовать прописным. Например, строчная "а" всегда будет идти перед прописной "А". Если соответствующий этой опции флажок не установлен, символы "а" и "А" считаются одинаковыми.
- Accent sensitive (Учитывать ударение). При выборе этой опции SQL Server 2000 различает ударные или безударные буквы (а также буквы с другими надстрочными знаками). Например, символы "а" и "á" рассматриваются как разные. Как и в предыдущем случае, если опция Accent sensitive не выбрана, такие буквы считаются одинаковыми.
- Kana sensitive (Учитывать разницу между алфавитами хирагана и катакана). При выборе этой опции SQL Server 2000 учитывает разницу между двумя алфавитами японского языка (хирагана и катакана). Если данная опция не выбрана, SQL Server рассматривает эти алфавиты как одинаковые.
- Width sensitive (Учитывать ширину символов в байтах). При выборе этой опции SQL Server 2000 учитывает разницу между однобайтовыми символами (половина длины символа) и соответствующими им двухбайтовыми символами (полная длина символа). Если данная опция не выбрана, эти символы считаются одинаковыми.

На заметку Бинарный способ сортировки может несколько отличаться от традиционного, согласно которому слова определенного языка располагаются в словаре. Поэтому люди, говорящие на этом языке, могут заметить, что слова отсортированы не совсем так, как они того ожидали.

Для рассматриваемой установки я рекомендую выбрать опцию Binary, однако только в том случае, если вам действительно не нужно, чтобы параметры настройки были совместимы с более ранними версиями SQL Server (об этом чуть ниже). Хотя в случае выбора опции Binary сортировка не всегда выполняется в том порядке, к которому вы привыкли, она происходит намного быстрее, чем все остальные типы сортировки, а, как известно, быстродействие — одно из неотъемлемых качеств хорошей базы данных.

И наконец, последняя опция в этом диалоговом окне называется SQL Collations (Параметры сортировки SQL). Как указано в скобках после названия опции, она используется для совместимости с предыдущими версиями SQL Server, включая 7.0, 6.5 и даже более старые. Выбирайте эту опцию только в том случае, если вашему экземпляру SQL Server придется взаимодействовать с экземплярами более ранних версий (для разработки приложения SQL Spy Net это не понадобится).

Нельзя также не упомянуть одну из наиболее примечательных новых возможностей SQL Server 2000 — поддержку сортировки на уровне базы данных. В предыдущих версиях SQL Server после настройки параметров сортировки изменить их было уже нельзя; для этого, увы, требовалось полностью переустановить систему. Однако в случае с SQL Server 2000 все гораздо проще: для сервера можно установить параметры сортировки Windows, а при создании базы данных установить для нее параметры сортировки SQL. Теперь во время установки сервера вам не нужно будет раскидывать карты или гадать на кофейной гуще, чтобы выяснить, какие же параметры сортировки могут понадобиться в будущем? Ну просто фантастика!

Когда все параметры сортировки будут настроены должным образом, щелкните на кнопке Next. Следующее диалоговое окно под названием Network Libraries (Сетевые библиотеки) показано на рис. Б.19.

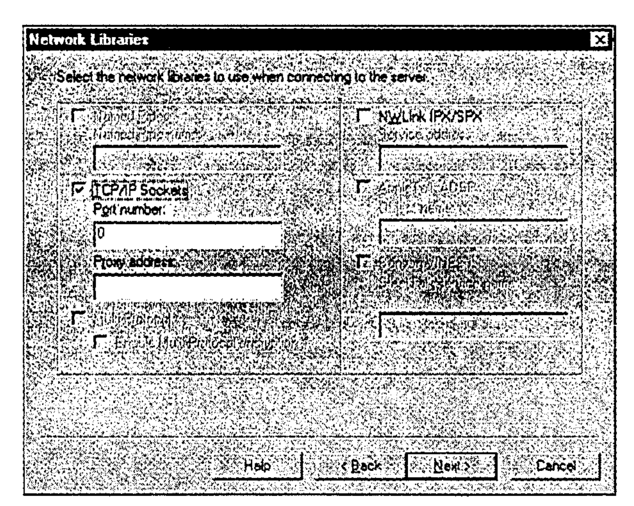


Рис. Б.19. Настройка сетевых библиотек при выборочной установке

Это диалоговое окно касается сетевых библиотек, которые следует установить для общения между *клиентом* и *сервером* в SQL Server 2000. Подробнее о том, как архитектура клиент/сервер связана с опциями этого диалогового окна, речь идет в главе 12, "Разработка интерфейса пользователя базы данных SQLSpyNet".

Термин *Сетевые библиотеки* (*network libraries*) — это коммуникационные протоколы, которые SQL Server 2000 использует для работы в различных сетях. Они позволяют SQL Server 2000 обмениваться информацией с другими компьютерами по сети путем передачи *пакетов*.

Термин *Пакеты* (*packets*) — это небольшие блоки информации, собранные для передачи по сети в единое целое. (Этот процесс аналогичен тому, как продавец, упаковывая товар для покупателя, кладет в пакет вместе с товаром чек и рекламное объявление.)

Экземпляр SQL Server 2000 может "прослушивать" сетевую библиотеку только в том случае, если она была должным образом настроена. Если же было настроено несколько сетевых библиотек, SQL Server 2000 может прослушивать каждую из них.

Для настройки коммуникационных протоколов SQL Server 2000 используются описанные ниже опции.

- Named Pipes. Используется только для настройки SQL Server 2000 под операционные системы Windows NT 4.0 или Windows 2000. Системы Windows 95/98 этот протокол не поддерживают.
- TCP/IP Sockets. Применяется по умолчанию; именно эту сетевую библиотеку используют все конфигурации SQL Server 2000, установленные под операционные системы, которые поддерживают данный протокол.
- Multi-Protocol. Применяется, если необходимо скомбинировать несколько сетевых протоколов, которые используются в вашей организации. К примеру, он поддерживает протоколы TCP/IP, NWLink, IPX/SPX и Named Pipes. Кроме того, Multi-Protocol разрешает применение идентификации Windows ко всем протоколам, которые поддерживаются средствами удаленного вызова процедуры Windows NT (Remote Procedure Call — RPC).
- NWLink IPX/SPX. Применяется при работе SQL Server 2000 в сети Novell и позволяет общаться с SQL Server 2000 клиентам, использующим протокол Novell SPX.
- AppleTalk ASDP. Позволяет клиентским компьютерам Apple Macintosh устанавливать соединение и обмениваться информацией с SQL Server 2000 с помощью протокола AppleTalk (а не TCP/IP).
- Banyan VINES. Еще один сетевой протокол, аналогичный хорошо известным протоколам сетей Novell и NT. Предназначен специально для сети VINES.

На заметку При установке именованного экземпляра (как это сделали мы) опции *Multi-Protocol*, *AppleTalk ADSP* и *Banyan VINES* будут недоступны.



В SQL Server 2000 протоколы AppleTalk ADSP и Banyan VINES не были усовершенствованы, а следовательно, остались на том же функциональном уровне, что и в версии SQL Server 7.0. Согласно документации Microsoft больше не будет заниматься усовершенствованием этих протоколов, а в дальнейших продуктах планирует отказаться от них совсем. Эти протоколы не поддерживаются именованными экземплярами.

Для нашей установки выберите опцию TCP/IP Sockets.

Выбор правильного порта

Экскурс

Как вы уже могли заметить, в поле *Port Number* (Номер порта) предыдущего диалогового окна у меня стоит ноль. Почему? Дело в том, что, как правило, для обмена информацией SQL Server использует порт 1433. Именно этот номер был выделен специально для SQL Server Агентством по выделению имени и уникальных параметров протоколов Internet (Internet Assigned Number Authority — IANA). Поскольку на моем компьютере уже установлен один экземпляр SQL Server 2000, я не могу опять использовать порт 1433, иначе он будет перекрыт новым экземпляром. Оставляя номер порта равным 0, SQL Server 2000 указывает на то, что номер порта будет выделяться динамически (тем самым позволяя нескольким экземплярам мирно уживаться на одном компьютере).

Если это первая установка SQL Server (независимо от версии) на данном компьютере, то в поле *Port Number* по умолчанию будет стоять 1433. В этом случае оставьте все как есть.

Тем не менее, если на вашем компьютере, как и на моем, уже имеется хотя бы один экземпляр SQL Server (любой версии), то в поле *Port Number* по умолчанию будет стоять 0, а поле для ввода адреса прокси-сервера останется пустым. В этом случае также оставьте все как есть.

Указывать номер порта необходимо только в случае, если вы желаете, чтобы SQL Server 2000 "слушал" порт, отличный от принятого по умолчанию порта 1433. Однако на компьютере с двумя и более именованными экземплярами SQL Server 2000 номер порта не назначается до тех пор, пока не запускается какой-либо экземпляр. Если именованному экземпляру назначен свой номер порта, при его запуске SQL Server 2000 будет использовать именно этот номер. И еще: перед назначением какого-нибудь порта, отличного от 1433, сначала убедитесь в том, что этот порт свободен!

Вот и все, что можно было сказать по поводу выборочной установки SQL Server 2000. Если теперь вы вернетесь к предыдущему разделу, а именно к началу процесса копирования файлов, и выполните указанные там шаги, все должно пойти как по маслу.

Проверка успешности установки

Ура! Ура! Ура! Установка SQL Server 2000 и средств управления наконец-то завершена!

В следующих разделах пойдет речь о том, как запустить SQL Server 2000 из меню Start (Пуск) и как проверить, работает ли вообще эта СУРБД на вашем компьютере.

После перезагрузки компьютера (если она была необходима) в меню Start (Пуск) появится новое подменю Programs⇨Microsoft SQL Server (Программы⇨Microsoft SQL Server), как показано на рис. Б.20.

Данное подменю содержит средства настройки и управления для SQL Server 2000. Чтобы проверить правильность работы экземпляра SQL Server 2000 на вашем компьютере, выберите в этом меню команду Service Manager (Диспетчер службы SQL). На экране появится диалоговое окно, изображенное на рис. Б.21.

В раскрывающемся списке Server (Сервер) перечислены экземпляры SQL Server 2000, установленные на вашем компьютере. В этом списке будет присутствовать и имя нового, только что установленного экземпляра, перед которым, возможно, будет стоять имя вашего компьютера, например *имя_компьютера\имя_экземпляра*.

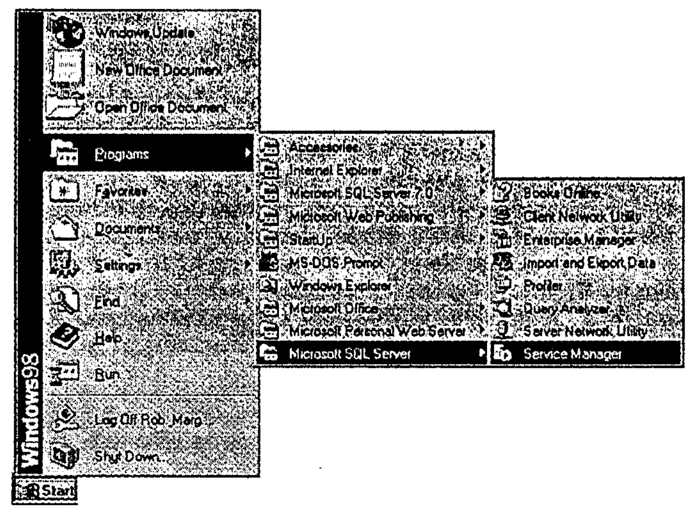


Рис. Б.20. Как найти группу программ SQL Server 2000 в меню Start

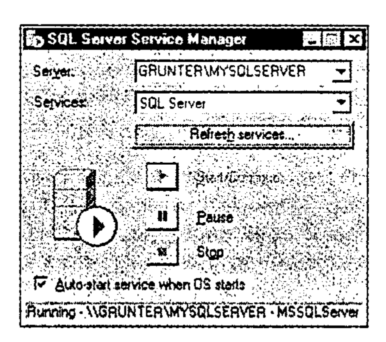


Рис. Б.21. Утилита Service Manager для SQL Server 2000

Выберите из списка экземпляр, который необходимо запустить. Затем, из следующего раскрывающегося списка выберите службу (а именно службу SQL Server), которую также необходимо запустить.

После этого щелкните на кнопке с зеленым треугольником, если она доступна. Это приведет к запуску службы SQL на вашем компьютере. Кроме того, установите флажок Auto-Start Service When OS Starts (Автоматически запускать службу при загрузке операционной системы) для того, чтобы служба SQL автоматически запускалась при включении компьютера.

Если же кнопка с зеленым треугольником недоступна, это значит, что служба SQL уже запущена. В этом случае остановите ее с помощью кнопки Stop (Остановить), а затем, щелкнув на кнопке с зеленым треугольником, запустите снова.

После этого убедитесь в том, что флажок Auto-Start Service When OS Starts установлен. Остановка и перезапуск службы SQL были необходимы для того, чтобы в дальнейшем она корректно запускалась при каждом включении компьютера.

Вот на этом и завершается наше руководство по установке SQL Server 2000. Довольно просто, не так ли? А теперь, после того как SQL Server 2000 успешно установлен, следует пройти несколько дополнительных этапов, чтобы настроить параметры, необходимые для установки вашего первого соединения.

Настройка SQL Server 2000

Как уже отмечалось, SQL Server 2000 имеет довольно понятный и дружелюбный пользовательский интерфейс, что позволяет быстро и легко освоиться с работой этой СУРБД. Наконец-то пришел ваш черед заняться этим! Я собираюсь познакомить вас с главным приложением в пакете программ SQL Server 2000 — с программой Enterprise Manager.

С помощью программы Enterprise Manager мы попытаемся подключиться к установленному ранее экземпляру SQL Server 2000. Надеюсь, вы еще не забыли свой пароль? Итак выполните следующее.

1. Выберите команду Start⇨Programs⇨Microsoft SQL Server⇨Enterprise Manager (Пуск⇨Программы⇨Microsoft SQL Server⇨Enterprise Manager), как показано на рис. Б.22.

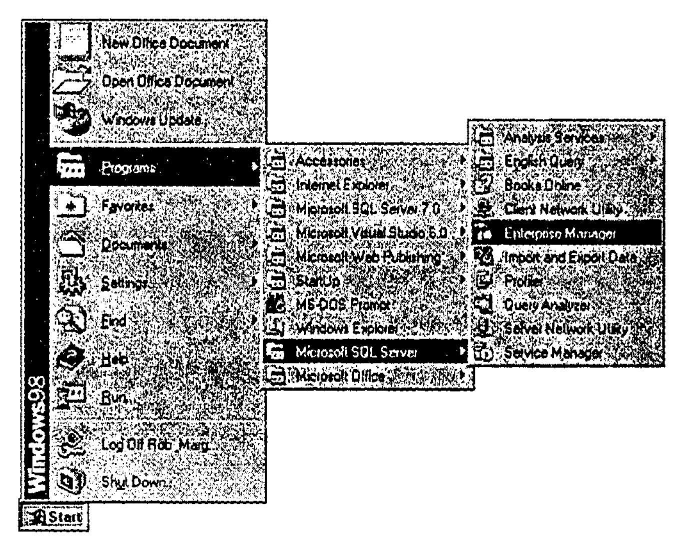


Рис. Б.22. Пиктограмма программы Enterprise Manager в меню Start

2. При первом запуске этой программы на экране появится окно консоли управления (Enterprise Management console), показанное на рис. Б.23.

Если вам действительно удалось забраться так далеко, значит, вы уже готовы установить соединение с SQL Server 2000 и продолжить разработку приложения.

Установка соединения с SQL Server 2000 в первый раз

Не знаю, как у вас, а у меня от этой фразы просто мурашки по коже. Почему? Ну как же, это ведь значит, что мы уже находимся на пути разработки нашего первого настоящего приложения в SQL Server 2000! Неужели вы не чувствуете приятного волнения? Разумеется, не все восторгаются одним и тем же, и это замечательно, иначе, на мой взгляд, мир был бы гораздо скучнее. В любом случае давайте-ка поскорее начнем!

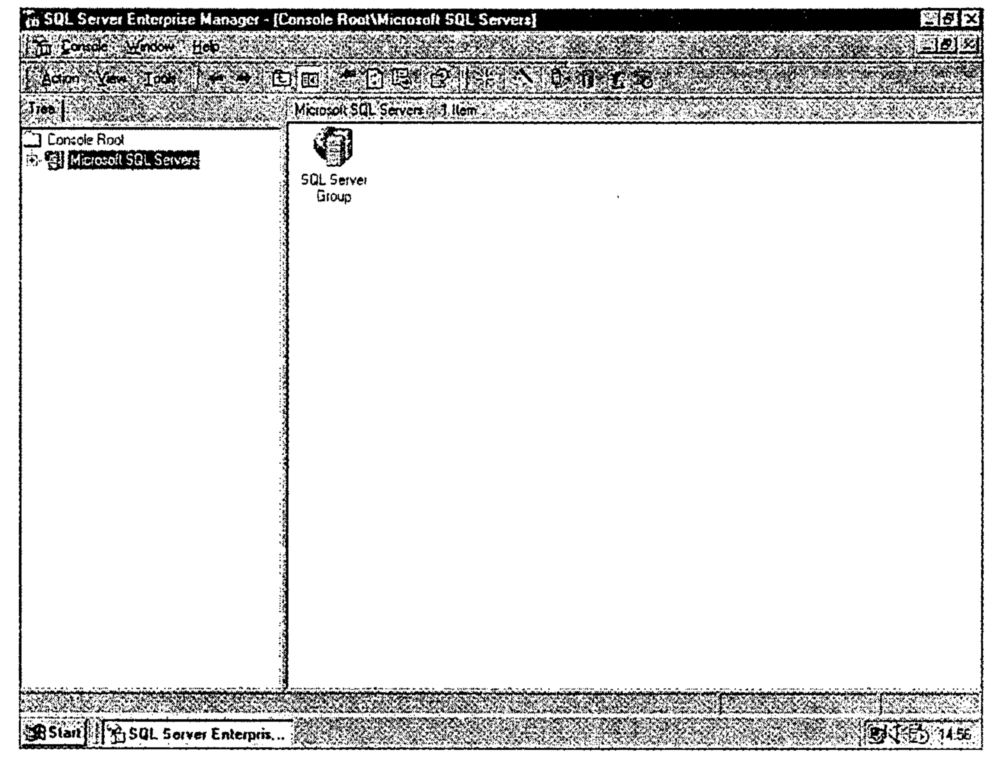


Рис. Б.23. Приложение Enterprise Manager для SQL Server 2000

1. Чтобы начать процесс установки соединения с SQL Server 2000, щелкните на знаке "+" возле пиктограммы Microsoft SQL Servers в верхней части дерева объектов, расположенного слева в окне консоли. Эта пиктограмма находится под пиктограммой папки Console Root (см. рис. Б.23).

   После раскрытия дерева на экране должен появиться новый элемент под названием SQL Server Group (Группа SQL Server). Если этого не случится, щелкните на пиктограмме SQL Servers правой кнопкой мыши и выберите из контекстного меню команду New SQL Server Group (Новая группа SQL Server), как показано на рис. Б.24.

2. В появившемся диалоговом окне введите название новой группы, как показано на рис. Б.25. Я назвал свою группу **SQL Server Group**, ну а вы можете назвать свою как захотите.

Созданной группе SQL Server будет принадлежать отдельный экземпляр или же кластер экземпляров SQL Server 2000. Это особенно полезно, когда у вас есть несколько серверов (или экземпляров), соответствующих разным версиям SQL Server. Например, у вас могут быть группы SQL Server 2000 и SQL Server 7.0, что позволит собрать вместе подобные между собой приложения.

После создания группы, которой будет принадлежать установленный вами экземпляр, необходимо создать реальное соединение с этим экземпляром. Для этого существует два пути (как, впрочем, и всегда в жизни, не так ли?). Вначале я расскажу вам об установке соединения вручную (поскольку предпочитаю именно этот способ), а затем об установке соединения с помощью мастера.

Рис. Б.24. Команда New SQL Server Group программы Enterprise Maneger

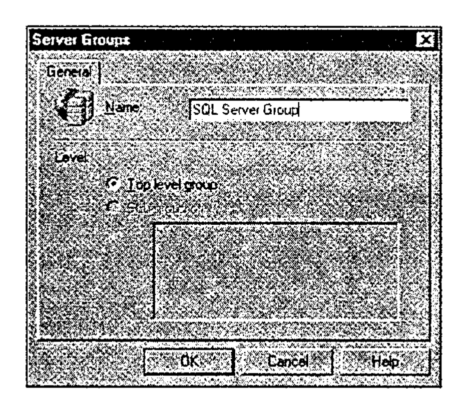


Рис. Б.25. Диалоговое окно регистрации групп SQL Server программы Enterprise Manager

1. Щелкните правой кнопкой мыши на пиктограмме SQL Server Group и выберите из появившегося контекстного меню команду New SQL Server Registration (Новая регистрация SQL Server), как показано на рис. Б.26.
2. После этого на экране должно появиться окно мастера регистрации SQL Server (Register SQL Server Wizard), показанное на рис. Б.27. Если этого не произойдет, перейдите к п. 3.

Рис. Б.26. Команда New SQL Server Registration приложения Enterprise Manager

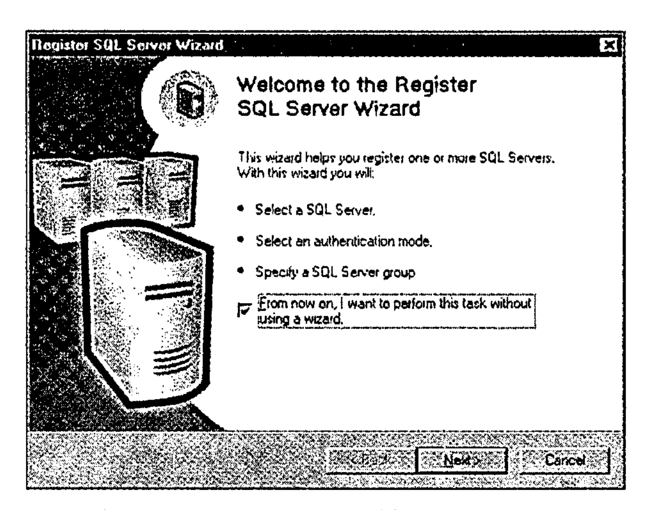


Рис. Б.27. Мастер регистрации SQL Server приложения Enterprise Manager

Чтобы продолжить установку соединения вручную, выберите опцию From Now On, I Want to Perform This Task Without Using a Wizard (С этого момента я хочу выполнить это задание без использования мастера регистрации), а затем щелкните на кнопке Next. На экране появится диалоговое окно, представленное на рис. Б.28.

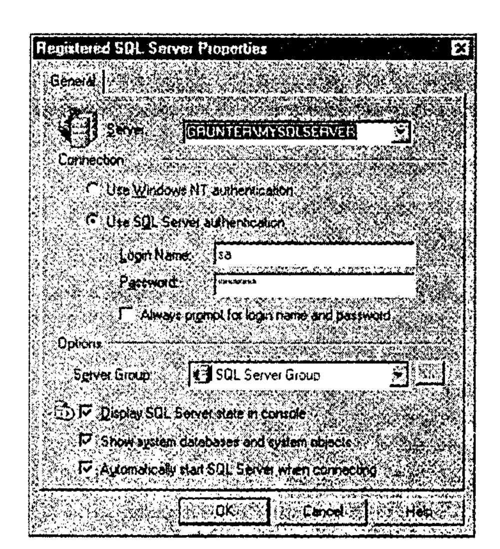


Рис. Б.28. Диалоговое окно регистрации SQL Server приложения Enterprise Manager

Если же вы хотите установить соединение с SQL Server 2000 с помощью мастера, не устанавливайте этот флажок, а просто щелкните на кнопке Next. Более подробно установка соединения с помощью мастера описана далее.

3. Данное диалоговое окно позволяет вручную задать информацию о соединении, необходимую для того, чтобы ваш компьютер смог общаться в качестве клиента с SQL Server 2000.

4. Первое поле этого диалогового окна позволяет ввести или же выбрать из списка имя экземпляра SQL Server. Если имя вашего экземпляра в этом списке отсутствует, просто введите его в формате *имя_сервера\имя_экземпляра*.

5. После этого следует ввести имя пользователя и пароль, которые вы будете использовать для подключения к SQL Server 2000. В поле Login Name (Имя пользователя) введите **sa**, а в поле Password (Пароль) — пароль учетной записи системного администратора, который вы назначили в процессе установки. Если же вы не назначали пароль, оставьте это поле пустым.

На заметку Если вы все-таки забыли свой пароль, это очень и очень плохо! Вы думали, я шучу? Нет, серьезно, в этом случае вам, к сожалению, придется удалить только что с такой любовью установленный экземпляр и установить новый. Это действительно очень неприятно, но что поделаешь? По крайней мере пароль вы больше никогда не забудете, это я вам гарантирую!

6. Если вы установите флажок Always Prompt for Login Name and Password (Всегда запрашивать имя пользователя и пароль), то при попытке подключения к SQL Server на экране всегда будет появляться окно регистрации пользователя. Это предохранит вас от несанкционированных попыток других пользователей установить соединение с вашим экземпляром SQL Server 2000.

   Если вы уверены, что с безопасностью вашего компьютера и так все в порядке, не выбирайте эту опцию.

7. Теперь необходимо назначить вашему соединению группу SQL Server, которой оно будет принадлежать. Выберем группу, созданную ранее в этом разделе. Если в раскрывающемся списке Server Group (Группа SQL Server) имени этой

группы нет, щелкните на кнопке с троеточием (...), расположенной справа от списка, и введите имя группы вручную. Убедитесь в том, что это группа верхнего уровня (это значит, что в дереве окна консоли Enterprise Manager она будет находиться на верхнем уровне, а не на подуровне какой-нибудь другой группы). Необходимая вам группа будет создана заново.

Последние три опции определяют вид и поведение SQL Server 2000 во время установки соединения.

- Display SQL Server state in console (Отображать состояние SQL Server в окне консоли). Включает и выключает статистику состояния SQL Server.
- Show system databases and system objects (Показывать системные базы данных и системные объекты). Подобно Windows, SQL Server использует некоторые системные объекты, такие как `tempDB`, `master` и т.д. Эта опция позволяет отображать подобные объекты в структуре дерева консоли.
- Automatically start SQL Server when connecting (Автоматически запускать SQL Server 2000 при установлении соединения). Указывает на то, что если SQL Server 2000 в данное время не запущен (работа службы SQL была остановлена), то всякий раз после установления соединения он будет перезапускаться автоматически.

Я предлагаю вам установить все три опции, если только у вас нет каких-либо своих соображений на этот счет.

Когда вы щелкнете на кнопке OK, Enterprise Manager попытается установить соединение с SQL Server 2000. Если подключение пройдет успешно (в зависимости от производительности компьютера это может занять от нескольких секунд до нескольких минут), все станет готово к просмотру содержимого SQL Server 2000. Окно Enterprise Manager при этом будет выглядеть примерно так, как показано на рис. Б.29.

На заметку

Если вы получили сообщение об ошибке, вначале проверьте правильность ввода имени пользователя и пароля, а затем правильность ввода комбинации *имя_сервера\имя_экземпляра*. Если и это не поможет, обратитесь к справочной системе Books Online (т.е. к поставляемой вместе с SQL Server 2000 документации). Она обеспечит вас всей необходимой информацией относительно возможных проблем установки. Кроме того, вы можете обратиться к Web-узлу компании Microsoft по адресу: `http://msdn.microsoft.com` или `http://www.microsoft.com/technet/` (эти адреса были действительны на момент написания книги; возможно, в будущем, они изменятся). Эти интерактивные службы окажут неоценимую поддержку в становлении вашей карьеры администратора баз данных, и, что самое приятное, всю информацию на них можно получить совершенно бесплатно!

8. И наконец, щелкните на знаке "+" рядом с пиктограммой *имя_сервера\имя_экземпляра* для того, чтобы просмотреть структуру SQL Server. После этого вы сможете видеть на экране все базы данных и другие сопутствующие объекты. На рис. Б.30 показано, как это примерно должно выглядеть.

Итак, соединение установлено! В следующем разделе пойдет речь о том, как подключиться к SQL Server 2000 с помощью мастера, хотя и без этого вы уже все знаете и умеете. За проявленные храбрость и терпение вам вполне можно поставить пять с плюсом. Вы просто молодец!

Рис. Б.29. Окно программы Enterprise Manager после установления соединения с SQL Server 2000

Рис. Б.30. Окно программы Enterprise Manager с развернутой структурой объектов баз данных

Использование мастера для установки соединения с SQL Server 2000

Итак, вы решили попытаться установить соединение с SQL Server 2000 с помощью встроенного мастера регистрации. После щелчка на кнопке Next в начальном диалоговом окне (об этом мы уже говорили) на экране появится диалоговое окно, показанное на рис. Б.31.

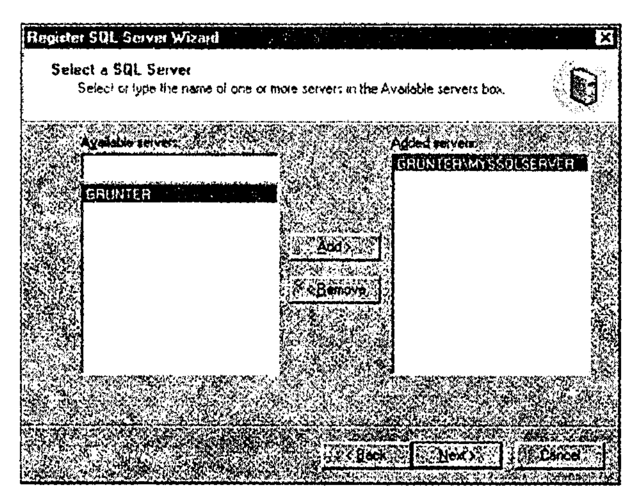


Рис. Б.31. Второе диалоговое окно мастера регистрации для приложения Enterprise Meneger

1. Если в списке Available Servers (Доступные серверы) нет имен ваших сервера и экземпляра, введите их вручную в формате *имя_сервера\имя_экземпляра.*



Чтобы узнать имя сервера, запустите приложение *SQL Service Manager* (Диспетчер службы SQL) с помощью команды меню *Start⇨Programs⇨Microsoft SQL Server⇨Service Manager* (Пуск⇨Программы⇨Microsoft SQL Server⇨Service Manager). В раскрывающемся списке *Server* главного окна этого приложения найдите необходимое вам имя экземпляра (оно уже будет в требуемом формате), скопируйте его и вставьте в поле *Available Servers.*

После выбора или ввода имени сервера щелкните на кнопке Add (Добавить). Имя сервера появится в списке Added servers (Добавленные серверы). Затем щелкните на кнопке Next.

2. Как показано на рис. Б.32, появившееся диалоговое окно позволяет задать тип идентификации, который Enterprise Manager будет использовать для подключения к SQL Server 2000.

   Выберите опцию The SQL Server login information that was assigned to me by the system administrator (Параметры идентификации SQL Server, которые были назначены мне системным администратором). Если ваш компьютер подсоединен к сети, вам, возможно, захочется выбрать опцию The Windows NT account information I use to log on my computer (Параметры идентификации Windows NT, которые я использую для входа в систему). Однако разрабатываемое приложение подразумевает использование системы безопасности SQL Server 2000, поэтому в данном случае следует выбрать именно первую опцию.

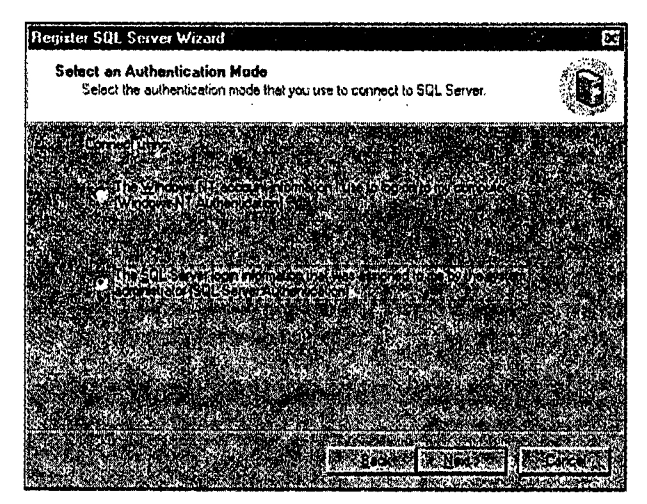


Рис. Б.32. Диалоговое окно выбора типа идентификации мастера регистрации для приложения Enterprise Manager

3. Щелкните на кнопке Next. Как показано на рис. Б.33, в следующем диалоговом окне следует ввести имя пользователя и пароль, которые вы будете использовать для подключения к SQL Server 2000.

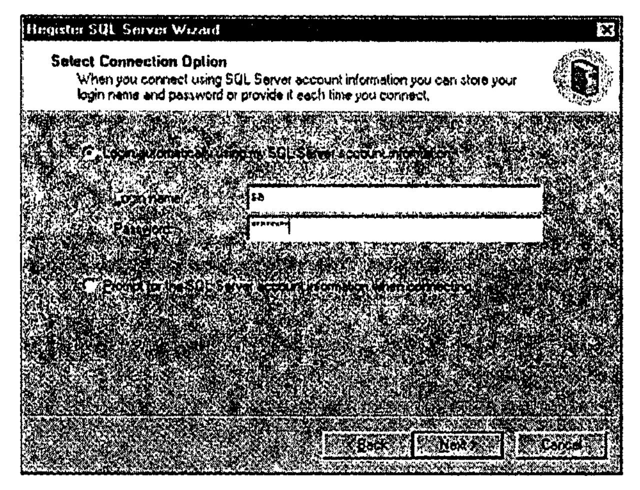
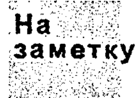


Рис. Б.33. Диалоговое окно для ввода имени пользователя и пароля системного администратора мастера регистрации Enterprise Meneger

В поле Login Name (Имя пользователя) введите **sa**, а в поле Password (Пароль) — пароль учетной записи системного администратора, который вы назначили в процессе установки. Если же вы не назначали пароль, оставьте это поле пустым.



Если вы все-таки забыли свой пароль, это очень и очень плохо! Вы думали, я шучу? Нет, серьезно, в этом случае вам, к сожалению, придется удалить только что с такой любовью установленный экземпляр и установить новый. Это действительно очень неприятно, но что поделаешь? По крайней мере пароль вы больше никогда не забудете, это я вам гарантирую.

Следующая опция под названием Prompt for the SQL Server account information when connecting (Всегда запрашивать имя пользователя и пароль при подключении) означает, что при попытке подключения к SQL Server с помощью каких-либо средств управления сервер всегда будет запрашивать имя пользователя и пароль. Это очень хорошо для совместно используемого компьютера, где необходимо ограничить доступ к SQL Server других пользователей. Однако если ваш компьютер и так защищен от несанкционированного доступа, выбирать эту опцию не следует. В любом случае щелкните на кнопке Next.

4. Следующее диалоговое окно, изображенное на рис. Б.34, позволяет назначить группу SQL Server, которой будет принадлежать ваше соединение. Выберите созданную нами ранее группу из раскрывающегося списка Group Name (Имя группы).

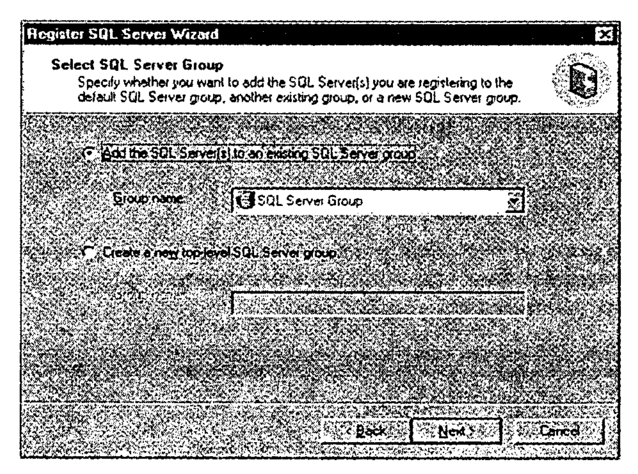


Рис. Б.34. Диалоговое окно выбора группы SQL Server



На заметку

Если в этом списке не оказалось имени вашей группы, выберите опцию *Create a new top-level SQL server group* (Создать новую группу SQL Server верхнего уровня) и введите желаемое имя группы. Необходимая группа будет создана заново. Затем щелкните на кнопке *Next*.

5. Последнее диалоговое окно (рис. Б.35) подтверждает выбранные вами имя сервера и имя экземпляра.

   Для внесения каких-либо изменений щелкните на кнопке Back (Назад). Для завершения работы мастера щелкните на кнопке Finish (Готово).

6. После этого на экране появится диалоговое окно, показанное на рис. Б.36. В нем отображается процесс установки соединения Enterprise Manager с экземпляром SQL Server 2000.

Теперь вы можете вернуться к п. 4 предыдущего раздела, и... как говорил Шекспир, "Живи, Мак-Дуфф!".

И как это все работает?

Если вы проводили выборочную установку SQL Server 2000 (если нет, вернитесь к началу этого приложения), то, должно быть, помните, как выполнялась настройка некоторых свойств и параметров сетевых библиотек. Эти библиотеки позволяют устанавливать соединения с SQL Server 2000.

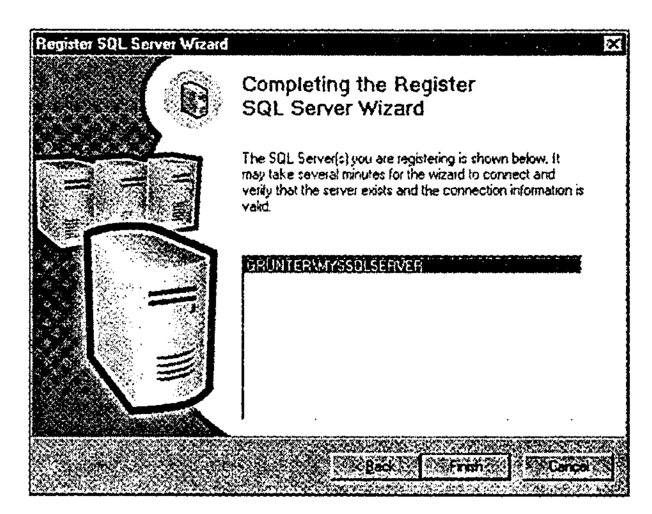


Рис. Б.35. Последнее диалоговое окно мастера регистрации

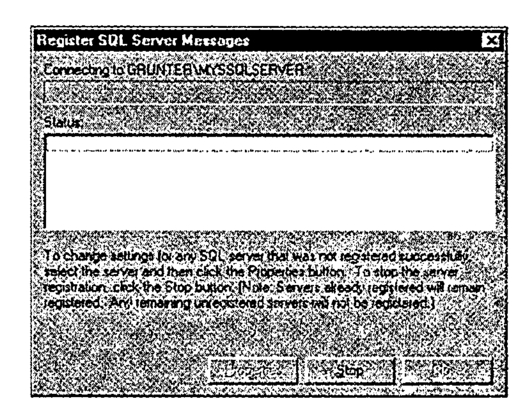


Рис. Б.36. Процесс регистрации и установки соединения

Поскольку мы устанавливаем SQL Server 2000 под управлением операционной системы Windows 98 и не подключены к сети, необходимо использовать сетевой протокол TCP/IP. Для того чтобы подключиться к SQL Server 2000, и клиент и сервер должны использовать общий протокол. По умолчанию в SQL Server 2000 устанавливаются и клиент и сервер, что позволяет избежать некоторых проблем соединения, с которыми вы бы неизбежно столкнулись, настраивая их вручную.

Вместе с SQL Server 2000 устанавливаются две утилиты, предназначенные для настройки протоколов, которые используются для связи с сервером: Server Network (Сетевые настройки сервера) и Client Network (Сетевые настройки клиента). Утилита Server Network используется для настройки протоколов, с помощью которых SQL Server 2000 будет "слышать" запросы клиента. А утилита Client Network используется для настройки сетевых библиотек, посредством которых клиент будет "разговаривать" с сервером.

Протокол TCP/IP — это все, что поддерживается экземпляром SQL Server 2000, установленным под управлением Windows 98. Это намного облегчает процесс настройки и обмена информацией с SQL Server 2000.

Устанавливая соединение с SQL Server 2000 с помощью программы Enterprise Manager, мы выступали в роли клиента. Мы формировали запрос к SQL Server 2000, а он отвечал на него, позволяя установить соединение. Таким образом, можно сказать, что SQL Server 2000 слушал наши запросы и удовлетворял их.

Я знаю, что кажется довольно странным, когда один и тот же компьютер является одновременно и сервером и клиентом. Тем не менее в реальной жизни подобных примеров не так уж мало. Вспомните, например, работу хозяина небольшого магазинчика. Стоя за прилавком, он выступает в роли сервера, а делающий заказы покупатель — в роли клиента. Однако, когда тот же хозяин магазина (сервер) заказывает товар у поставщика, он становится клиентом.

Конечно, это всего лишь краткий обзор, но по мере изучения материала книги вы сможете узнать об архитектуре клиент/сервер побольше. Поэтому давайте-ка вернемся к разработке нашего приложения SQL Spy Net.

Теперь для разработки приложения все готово, поэтому хватит сидеть сложа руки! Пора переходить к делу!

Приложение В

Ну и что дальше?

В этом приложении...

Ну что ж, ребятки, как это ни грустно и ни удивительно, но наша книга наконец-то подходит к концу. Я получил огромное удовольствие от этого грандиозного проекта и искренне надеюсь, что вы тоже. Как видите, мы, новозеландцы, не такие уж и нудные!

В этом приложении вам предлагается освоить некоторые возможности, открывшиеся перед вами как только вы стали обладателем обширнейших новых знаний. Кроме того, вы узнаете, как закрепить и усовершенствовать эти знания путем дальнейшего обучения и сертификации.

Как вы уже, наверное, догадались, в должности администратора баз данных скучать не приходится. Вас никогда не оставляет мысль о том, что, придя сегодня утром на работу, вы обязательно столкнетесь с чем-то новым и неожиданным, будь то исправление неполадок в базе данных клиента, у которого что-то там не сложилось, или же поиск нового способа выполнения рутинных каждодневных задач.

Если вы не мыслите себя и свою работу без постоянных приключений, тогда, возможно, должность администратора баз данных подойдет вам как нельзя лучше. Кроме того, сейчас за хорошего администратора баз данных компании готовы выложить огромные деньги, поэтому зарплату вы практически назначаете себе сами. Здорово, правда?

Администраторы баз данных SQL сегодня требуются во многих организациях по всему миру. Я перечислю несколько Web-узлов, которые могут помочь вам в поиске работы (но только не подумайте, что я имею к ним какое-либо отношение!). Просто, на мой взгляд, эти службы предлагают наибольшее количество вакансий администраторов баз данных (по крайней мере, так было к моменту завершения работы над книгой).

Один из наиболее приятных моментов состоит в том, что для работы в области информационных технологий вовсе не обязательно иметь диплом учебного заведения, как, скажем, юристам или бухгалтерам. Практический опыт, готовность к самообучению и желание идти вперед — вот все, что вам нужно.

Пожалуйста, не поймите меня неправильно. Если у вас есть возможность поступить в университет и получить диплом, не теряйте ее! Хотя обучение в университете носит в основном теоретический характер, приобретенные там знания позволят разобраться в таких довольно сложных для самостоятельного изучения концепциях, как:

- теория реляционных баз данных;
- жизненный цикл программного обеспечения (Software Development Life Cycle — SDLC);
- эффективное создание баз данных;
- дисковое пространство и его оптимизация;
- многопользовательские среды, включая блокировку и транзакции.

Тем не менее, вне зависимости от того, есть ли у вас соответствующее образование или нет, Microsoft предлагает получение целого ряда профессиональных квалификаций, которые признаются во всем мире и могут способствовать продвижению вашей карьеры. Получение квалификации в Microsoft состоит из прохождения нескольких тестов.

О них-то мы сейчас и поговорим.

Сертификация Microsoft

Компания Microsoft предлагает целый ряд квалификационных экзаменов для прохождения сертификации.

Как и следовало ожидать, эти экзамены в основном касаются технологий Microsoft. Кроме того, они включают в себя некоторые принципы разработки, применяемые в Microsoft, например Microsoft Solution Framework (MSF), описывающие распределение ролей между участниками процесса разработки приложений, а также итерации (циклы), которые проходит приложение на протяжении своего жизненного цикла.

Вы можете сдать экзамены практически по каждому продукту Microsoft, включая следующие (но отнюдь не ограничиваясь ими):

- Visual Basic;
- SQL Server;
- Windows 2000 Professional;
- Windows 2000 Server;
- Office 2000.

Поэтому в какой бы области вы не чувствовали себя уверенно, у Microsoft всегда припасена пара-тройка экзаменов, которые помогут вам понять, насколько ваша уверенность обоснованна.

Какие же типы сертификации существуют и что требуется для прохождения каждой из них? В технической области (т.е. в сфере разработки) существует пять основных квалификаций.

На заметку Существуют и другие сертификаты Microsoft, например MOUS, но я ограничусь рассмотрением только основных, касающихся сферы разработки.

Компанией Microsoft создан специальный Web-узел, посвященный обучению и разработке. В конце каждого раздела я еще буду говорить о том, где именно можно получить более подробные сведения о той или иной сертификации, но адрес главной страницы во всех случаях один и тот же: `http://www.microsoft.com/mcp/`.

Ну а теперь перейдем непосредственно к сертификатам, предлагаемым Microsoft.

Как стать сертифицированным специалистом Microsoft (Microsoft-Certified Professional – MCP)

Это первый сертификат, который можно (и не так уж сложно) получить. Все, что для этого требуется, — успешно сдать *один* из технических экзаменов Microsoft. Звучит довольно просто, не так ли? В принципе это действительно так.

Экзамены, проводимые компанией Microsoft, — довольно серьезное испытание. Однако, соответствующим образом подготовившись и приложив определенные усилия, вы сможете избежать проблем с их сдачей.

Экзамены охватывают различные сферы экспертных знаний, включая разработку (Visual Basic, Visual C++ и т.д.), BackOffice (Windows 2000 Professional и Server), Internet (TCP/IP), Web-дизайн (Visual InterDev, FrontPage) и другие области.

Более подробную информацию о требованиях, необходимых для прохождения сертификации MCP, можно получить, выбрав одноименную команду меню, которое находится в левой части окна главной страницы сертификации Microsoft.

Кроме MCP, существует еще два дополнительных сертификата.

- MCP + Site Building (MCP + Web-дизайн). Для получения этого сертификата необходимо сдать два специализированных экзамена Microsoft, отвечающих требованиям к квалификации специалиста по Web-дизайну. После сдачи этих экзаменов вы получите не только сертификат MCP + Site Building, но и просто MCP. Как вам это?
- MCP + Internet. Эта сертификация похожа на предыдущую за исключением того, что для ее прохождения необходимо сдать три экзамена, отвечающих требованиям к квалификации специалиста по Internet. Как и в первом случае, после успешной сдачи экзамена вы получите не только сертификат MCP + Internet, но и просто MCP.

Более подробно об этих сертификациях можно узнать, выбрав команды меню MCP + Site Builder и MCP + Internet в левой части главной страницы сертификации Microsoft.

Как стать сертифицированным разработчиком решений Microsoft (Microsoft-Certified Solution Developer – MCSD)

Требования для получения этого сертификата выше, чем для получения сертификата MCP. Для прохождения сертификации MCSD необходимо сдать четыре экзамена, касающихся разработки программного обеспечения и ее этапов. Три экзамена обязательны, но в пределах каждого из них допускается некоторый выбор. Так, например, в качестве одного из обязательных вам предлагается выбрать экзамен по Visual Basic или же по Visual C++.

Четвертый экзамен считается выборочным. Здесь вам наконец-то позволено выбирать из довольно большого списка предметов, включающего и SQL Server.

Более подробную информацию об этом виде сертификации можно получить, выбрав команду меню MCSD в левой части окна главной страницы сертификации Microsoft.

Сертификат MCSD не предполагает получения каких-либо дополнительных сертификатов. Тем не менее после сдачи одного из экзаменов, необходимых для прохождения сертификации MCSD, вы автоматически получаете сертификат MCP. Таким образом, стать квалифицированным специалистом нетрудно и в этом случае.

Как стать сертифицированным системным инженером Microsoft (Microsoft-Certified Systems Engineer – MCSE)

Требования к прохождению этой сертификации, пожалуй, наиболее высоки. Семь — да, да, именно столько — экзаменов необходимо сдать для получения сертификата MCSE.

На заметку Все, о чем я здесь рассказыааю, касается работы с Windows 2000. На сегодня это наиболее перспективный путь сертификации, поскольку большинство экзаменов по Windows NT 3.51 вскоре будут свернуты, а в недалеком будущем такая же участь ожидает и экзамены по Windows NT 4.0.

Из этих семи экзаменов четыре являются обязательными (и выбора в пределах каждого из них практически нет). В качестве пятого экзамена на выбор предлагается уже два. К счастью, шестой и седьмой экзамены выборочные, и, как и в предыдущем случае, здесь вам позволено выбирать из довольно большого списка предметов.

Более подробно о сертификации MCSE можно узнать, выбрав команду меню MCSE в левой части окна главной страницы сертификации Microsoft.

Как и в предыдущих случаях, сдача одного из экзаменов автоматически подразумевает получение сертификата MCP.

В пределах сертификации MCSE возможно получение дополнительного сертификата под названием MCSE + Internet. Если у вас хватит смелости принять и этот вызов, вам придется сдавать целых *девять* экзаменов, семь из которых являются обязательными, а два — выборочными.

Более подробно об этом виде сертификации можно узнать, выбрав соответствующий пункт меню, расположенного в левой части окна главной страницы сертификации Microsoft.

Вам кажется, что девять экзаменов — это слишком тяжело? Да нет, на самом деле все не так уж и страшно. При правильной организации подготовки к экзаменам и наличии некоторой самодисциплины получить сертификат MCSE + Internet можно за полгода, в крайнем случае за год (и это, как говорится, без отрыва от производства!).

Как стать сертифицированным администратором баз данных Microsoft (Microsoft-Certified Database Administrator – MCDBA)

Я специально оставил этот тип сертификации напоследок, потому что именно он имеет самое непосредственное отношение к тому, о чем шла речь в этой книге!

Для получения сертификата MCDBA необходимо сдать *пять* экзаменов.

На заметку Как и в предыдущем случае, сертификация, о которой я говорю, касается работы с Windows 2000.

Четыре из пяти экзаменов обязательны, и выбор в пределах каждого из них не так уж велик. Последний экзамен, как вы, наверное, уже догадались, является выбороч-

ным и предлагает обширный список предметов. Как и в предыдущих случаях, сдача одного из экзаменов подразумевает автоматическое получение сертификата MCP.

Чтобы получить более подробную информацию об этом типе сертификации, выберите команду меню MCDBA в левой части окна главной страницы сертификации Microsoft.

Мы ознакомились с основными типами сертификатов. А как же лучше выбрать экзамены для получения каждого из них?

Как распланировать прохождение сертификации Microsoft

Для различных типов сертификаций многие экзамены совпадают, поэтому экзамен, который успешно сдан в рамках одной сертификации, может быть засчитан еще раз при прохождении другой! Это похоже на университет, где при одновременном обучении по нескольким специальностям экзамены по общим предметам могут быть"перезачтены".

Перед выбором экзамена хорошенько проанализируйте, какие экзамены могут быть перезачтены в рамках наибольшего количества сертификаций. Для этого необходимо изучить требования к прохождению каждой сертификации (как обязательные, так и экзамены по выбору).



Не выбирайте экзамены, наиболее полезные с точки зрения "перезачетов", но не отвечающие вашим профессиональным интересам. Постарайтесь найти баланс между получением как можно большего количества сертификатов и продвижением карьеры именно в той области, которая вас интересует.

Для прохождения новой сертификации нет необходимости специально обращаться в Microsoft; новый сертификат выдается автоматически, как только у вас набирается достаточное для его получения количество экзаменов.

В связи с этим возникает естественный вопрос: "А как я узнаю, что уже готов к сдаче экзамена?" К сожалению, ответа на этот вопрос не существует; вы должны почувствовать это сами. Существуют учебные пособия, которые могут помочь вам оценить свои возможности и навыки, и мы о них еще поговорим, но в конечном счете все зависит от вас.

Тем не менее, если вы провалили экзамен, это вовсе не смертельно. Да, конечно, в сертификате может быть записано, что вы не сдали этот экзамен с первого раза, но в конце концов, если у вас сертификат все-таки есть, а у других его нет, то какая уже разница?

Вы, конечно же, помните, что каждое из средств Microsoft заключает в себе тысячи возможностей, и хорошо знать их *все* практически невозможно. Но поскольку на экзамене вам может попасться все что угодно, постарайтесь разобраться в работе как можно большего числа средств данного продукта. Например, для сдачи экзамена по SQL Server постарайтесь получше изучить работу таких средств, как Enterprise Manager, Query Analyzer, Profiler, сетевых служебных приложений клиента и сервера и т.д.

В следующем разделе мы поговорим об учебных пособиях, предназначенных для подготовки к сертификации.

Учебные пособия

Один из наиболее распространенных способов подготовки к экзамену — получение практических навыков работы с продуктом.

К каждому экзамену компанией Microsoft ставится несколько обязательных требований, включая опыт работы экзаменуемого с данным продуктом или средством. Тем не менее в качестве подготовки Microsoft предлагает набор практических электронных тестов, которые позволят вам понять, что же представляет собой тот или иной экзамен.



Эти тесты предоставляют всего лишь возможность попрактиковаться в сдаче экзамена. Они разработаны для того, чтобы вы получили определенное понятие о том, как выглядят электронные экзамены. Результаты тестов ни в коей мере не отображают того, что вы получите на настоящих экзаменах.

Кроме того, при подготовке к экзаменам вам пригодятся книги и учебные пособия издательства Microsoft Press, а также других издателей.

Между тем Microsoft — отнюдь не единственная компания, способная помочь вам в этом нелегком деле. Целый ряд других компаний, например Transcender (`www.transcender.com`), предлагают тесты, имитирующие настоящие экзамены Microsoft.

Компания Transcender предлагает два типа подобных программ: это TranscenderCert и TranscenderFlash. Программа TranscenderCert имитирует проведение экзамена, а программа TranscenderFlash предлагает обзор основных вопросов, касающихся темы предстоящего экзамена, в виде электронных карточек с текстом и картинками.

Множество полезных советов относительно сдачи экзаменов можно получить и на многих других Web-узлах.

Итак, вот мой последний совет по поводу экзаменов: не бойтесь! Попробуйте свои силы! Даже если вы не сдадите экзамен с первого раза, вы по крайней мере приобретете кое-какой опыт и будете лучше подготовлены к следующей сдаче!

Возможности получения работы

А вот и то, чего вы, возможно, ждали на протяжении всей книги! Где же можно применить полученные знания?

Как уже отмечалось, хороших администраторов баз данных SQL нынче, что называется, и днем с огнем не сыщешь. Если вы действительно хороший администратор баз данных и знаете свою работу, вам цены нет. Но как же к этому прийти?

К сожалению, все начинают с самого низа. Чтобы попасть на работу, вам прежде всего придется показать, что вы напористы, целеустремленны и готовы к обучению.



К сожалению, я не профессиональный консультант службы занятости. Для получения действительно профессиональной помощи обращайтесь к соответствующим людям.

После получения должности помощника администратора баз данных вам, скорее всего, поначалу придется выполнять довольно банальные задания вроде преобразования данных из предыдущих систем и сопровождения существующих. Не дайте этим скучным заданиям отбить у вас охоту к тому, чем вы действительно хотите заниматься! Без опыта, который вам помогут приобрести такие, казалось бы, "мирские заботы", вы не сможете эффективно анализировать работу сервера и предупреждать потенциальные проблемы. А без досконального знания структур данных вы будете не в состоянии разработать новое, эффективное приложение.

Ну и наконец, множество Web-узлов помогут вам в поиске работы, главной составляющей которой было бы именно использование SQL Server.

Целый ряд Web-узлов предлагает бесплатное интерактивное обучение, поэтому, прежде чем обратиться к консультанту из агентства по найму, проверьте и эту возможность.



Еще несколько советов, которые могут пригодиться в поисках работы.

Непрямой путь: пусть поисками работы вместо вас занимается кадровое агентство. Найдите место, куда можно поместить свое резюме, или отдайте его в какое-нибудь солидное агентство по найму. Найти агентство, специализирующееся именно в той области, которая вас интересует, можно с помощью крупных Web-узлов типа Monster. Когда вы найдете объявление, похожее на то, что вы ищете, свяжитесь с давшим его кадровым агентством. Однако будьте осторожны и не продавайте себя кому попало.

Прямой путь: посетите Web-узлы приглянувшихся вам компаний и поищите на них информацию о работе. Большинство компаний всегда имеет в запасе несколько свободных вакансий, привлекая специалистов из различных отраслей.

Создайте собственный Web-узел: кстати, если вы действительно можете выполнить все, что описано в этой книге, создайте собственное приложение SQL Server 2000 для Web-узла, продемонстрировав свои способности всему миру. Или, по крайней мере, просто создайте собственный Web-узел, содержащий всю информацию о ваших навыках и способностях, которая могла бы заинтересовать потенциального работодателя.

Если вам понадобится дальнейшая помощь в поисках работы для начала карьеры, свяжитесь с местным кадровым агентством, специализирующимся в области информационных технологий; сегодня их развелось великое множество. Они смогут порекомендовать вам то, что действительно поможет найти желанную и любимую работу!

Спасибо тебе, читатель!

Поскольку книга, похоже, действительно подошла к концу, мне хотелось бы выразить самую искреннюю признательность за то, что вы нашли время, чтобы прочитать ее.

Конечно же, я понимаю, что затронул всего лишь верхушку огромного айсберга под названием SQL Server 2000, но искренне надеюсь, что действительно обеспечил вас всем необходимым для разработки и воплощения в жизнь своего первого ориентированного на использование базы данных SQL Server 2000 приложения.

Еще раз, мои дорогие читатели, спасибо вам за проявленные терпение и снисходительность к моему извращенному чувству юмора.

Возможно, когда-нибудь на ваших журнальных столиках появится еще одна моя книга, и, вполне вероятно, она будет посвящена очередной версии SQL Server! Подозреваю, однако, что случится это не скоро... Однажды мне сказали: написать книгу — это все равно, что "родить" собственные мысли. Если это правда, тогда сей труд окажется воистину долгим и тяжелым!

В любом случае постарайтесь еще раз вспомнить все то, о чем мы с вами говорили, а при необходимости вернитесь к первой странице и начните все сначала.

Спасибо тебе, читатель!

Ваш благодарный учитель
Роб Хоторн (Rob Hawthorne)

Предметный указатель

Л

М

Н

О

П

Р

У

Ф

Х

Ц

Ш

Э

Я